

A Boyer-Moore Type Algorithm for Timed Pattern Matching

Masaki Waga¹, Takumi Akazaki^{1,2}, and Ichiro Hasuo¹

¹ University of Tokyo, Tokyo, Japan

² JSPS Research Fellow, Tokyo, Japan

Abstract. The *timed pattern matching* problem is formulated by Ulus et al. and has been actively studied since, with its evident application in monitoring real-time systems. The problem takes as input a *timed word/signal* and a *timed pattern* (specified either by a *timed regular expression* or by a *timed automaton*); and it returns the set of those intervals for which the given timed word, when restricted to the interval, matches the given pattern. We contribute a *Boyer-Moore* type optimization in timed pattern matching, relying on the classic Boyer-Moore string matching algorithm and its extension to (untimed) pattern matching by Watson and Watson. We assess its effect through experiments; for some problem instances our Boyer-Moore type optimization achieves speed-up by two times, indicating its potential in real-world monitoring tasks where data sets tend to be massive.

1 Introduction

Importance of systems' *real-time* properties is ever growing, with rapidly diversifying applications of computer systems—cyber-physical systems, health-care systems, automated trading, etc.—being increasingly pervasive in every human activity. For real-time properties, besides classic problems in theoretical computer science such as *verification* and *synthesis*, the problem of *monitoring* already turns out to be challenging. Monitoring asks, given an execution log and a specification, whether the log satisfies the specification; sometimes we are furthermore interested in *which segment* of the log satisfies/violates the specification. In practical deployment scenarios where we would deal with a number of very long logs, finding matching segments in a computationally tractable manner is therefore a pressing yet challenging matter.

In this context, inspired by the problems of *string* and *pattern matching* of long research histories, Ulus et al. recently formulated the problem of *timed pattern matching* [22]. In their formalization, the problem takes as input a *timed signal* w (values that change over the continuous notion of time) and a *timed regular expression (TRE)* \mathcal{R} (a real-time extension of regular expressions); and it returns the *match set* $\mathcal{M}(w, \mathcal{R}) = \{(t, t') \mid t < t', w|_{(t, t')} \in L(\mathcal{R})\}$, where $w|_{(t, t')}$ is the restriction of w to the time interval (t, t') and $L(\mathcal{R})$ is the set of signals that match \mathcal{R} .

Since its formulation timed pattern matching has been actively studied. The first offline algorithm is introduced in [22]; its application in conditional performance evaluation is pursued in [12]; and in [23] an online algorithm is introduced based on *Brzozowski derivatives*. Underlying these developments is the fundamental observation [22]

that the match set $\mathcal{M}(w, \mathcal{R})$ —an uncountable subset of $\mathbb{R}_{\geq 0}^2$ —allows a finitary symbolic representation by inequalities.

Contributions In this paper we are concerned with *efficiency* in timed pattern matching, motivated by our collaboration with the automotive industry on various lightweight verification techniques. Towards that goal we introduce optimization that extends the classic *Boyer-Moore* algorithm for string matching (finding a pattern string *pat* in a given word *w*). Specifically we rely on the extension of the latter to *pattern matching* (finding subwords of *w* that is accepted by an NFA \mathcal{A}) by Watson & Watson [25], and introduce its *timed* extension.

We evaluate its efficiency through a series of experiments; in some cases (including an automotive example) our Boyer-Moore type algorithm outperforms a naive algorithm (without the optimization) by twice. This constant speed-up may be uninteresting from the complexity theory point of view. However, given that in real-world monitoring scenarios the input set of words *w* can be literally *big data*,³ halving the processing time is a substantial benefit, we believe.

Our technical contributions are concretely as follows: 1) a (naive) algorithm for timed pattern matching (§4); 2) its online variant (§4); 3) a proof that the match set allows a finitary presentation (Thm. 4.3), much like in [22]; and 4) an algorithm with Boyer-Moore type optimization (§5). Throughout the paper we let (timed) patterns expressed as *timed automata* (TA), unlike timed regular expressions (TRE) in [12, 22, 23]. Besides TA is known to be strictly more expressive than TRE (see [14] and also Case 2 of §6), our principal reason for choosing TA is so that the Boyer-Moore type pattern matching algorithm in [25] smoothly extends.

Related and Future Work The context of the current work is *run-time verification* and *monitoring* of cyber-physical systems, a field of growing research activities (see e.g. recent [13, 16]). One promising application is in *conditional quantitative analysis* [12], e.g. of fuel consumption of a car during acceleration, from a large data set of driving record. Here our results can be used to efficiently isolate the acceleration phases.

Aside from timed automata and TREs, *metric* and *signal temporal logics* (MTL/STL) are commonly used for specifying continuous-time signals. Monitoring against these formalisms has been actively studied, too [9–11, 15]. It is known that an MTL formula can be translated to a timed alternating automaton [20]. MTL/STL tend to be used against “smooth” signals whose changes are continuous, however, and it is not clear how our current results (on timed-stamped finite words) would apply to such a situation. One possible practical approach would be to quantize continuous-time signals.

Being *online*—to process a long timed word *w* one can already start with its prefix—is obviously a big advantage in monitoring algorithms. In [23] an online timed pattern matching algorithm (where a specification is a TRE) is given, relying on the timed extension of *Brzowski derivative*. We shall aim at an online version of our Boyer-Moore type algorithm (our online algorithm in §4 is without the Boyer-Moore type optimization), although it seems hard already for the prototype problem of string matching.

³ For example, in [8], a payment transaction record of 300K users over almost a year is monitored—against various properties, some of them timed and others not—and they report the task took hundreds of hours.

It was suggested by multiple reviewers that use of *zone automata* can further enhance our Boyer-Moore type algorithm for timed pattern matching. See Rem. 5.6.

Organization of the Paper We introduce necessary backgrounds in §2, on: the basic theory of timed automata, and the previous Boyer-Moore algorithms (for string matching, and the one in [25] for (untimed) pattern matching). The latter will pave the way to our main contribution of the timed Boyer-Moore algorithm. We formulate the timed pattern matching problem in §3; and a (naive) algorithm is presented in §4 together with its online variant. In §5 a Boyer-Moore algorithm for timed pattern matching is described, drawing intuitions from the untimed one and emphasizing where are the differences. In §6 we present the experiment results; they indicate the potential of the proposed algorithm in real-world monitoring applications.

Most proofs are deferred to the appendix due to lack of space.

2 Preliminaries

2.1 Timed Automata

Here we follow [1, 3], possibly with a fix to accept finite words instead of infinite. For a sequence $\bar{s} = s_1 s_2 \dots s_n$ we write $|\bar{s}| = n$; and for i, j such that $1 \leq i \leq j \leq |\bar{s}|$, $\bar{s}(i)$ denotes the element s_i and $\bar{s}(i, j)$ denotes the subsequence $s_i s_{i+1} \dots s_j$.

Definition 2.1 (timed word) A *timed word* over an alphabet Σ is an element of $(\Sigma \times \mathbb{R}_{>0})^*$ —which is denoted by $(\bar{a}, \bar{\tau})$ using $\bar{a} \in \Sigma^*$, $\bar{\tau} \in (\mathbb{R}_{>0})^*$ via the embedding $(\Sigma \times \mathbb{R}_{>0})^* \hookrightarrow \Sigma^* \times (\mathbb{R}_{>0})^*$ —such that for any $i \in [1, |\bar{\tau}| - 1]$ we have $0 < \tau_i < \tau_{i+1}$. Let $(\bar{a}, \bar{\tau})$ be a timed word and $t \in \mathbb{R}$ be such that $-\tau_1 < t$. The *t-shift* $(\bar{a}, \bar{\tau}) + t$ of $(\bar{a}, \bar{\tau})$ is the timed word $(\bar{a}, \bar{\tau} + t)$, where $\bar{\tau} + t$ is the sequence $\tau_1 + t, \tau_2 + t, \dots, \tau_{|\bar{\tau}|} + t$. Let $(\bar{a}, \bar{\tau})$ and $(\bar{a}', \bar{\tau}')$ be timed words over Σ such that $\tau_{|\bar{\tau}|} < \tau'_1$. Their *absorbing concatenation* $(\bar{a}, \bar{\tau}) \circ (\bar{a}', \bar{\tau}')$ is defined by $(\bar{a}, \bar{\tau}) \circ (\bar{a}', \bar{\tau}') = (\bar{a} \circ \bar{a}', \bar{\tau} \circ \bar{\tau}')$, where $\bar{a} \circ \bar{a}'$ and $\bar{\tau} \circ \bar{\tau}'$ denote (usual) concatenation of sequences over Σ and $\mathbb{R}_{>0}$, respectively. Now let $(\bar{a}, \bar{\tau})$ and $(\bar{a}'', \bar{\tau}'')$ be arbitrary timed words over Σ . Their *non-absorbing concatenation* $(\bar{a}, \bar{\tau}) \cdot (\bar{a}'', \bar{\tau}'')$ is defined by $(\bar{a}, \bar{\tau}) \cdot (\bar{a}'', \bar{\tau}'') = (\bar{a}, \bar{\tau}) \circ ((\bar{a}'', \bar{\tau}'') + \tau_{|\bar{\tau}|})$. A *timed language* over an alphabet Σ is a set of timed words over Σ .

Remark 2.2 (signal) *Signal* is another formalization of records with a notion of time, used e.g. in [22]; a signal over Σ is a function $\mathbb{R}_{\geq 0} \rightarrow \Sigma$. A timed word describes a time-stamped sequence of events, while a signal describes values of Σ that change over time. In this paper we shall work with timed words. This is for technical reasons and not important from the applicational point of view: when we restrict to those signals which exhibit only finitely many changes, there is a natural correspondence between such signals and timed words.

Let C be a (fixed) finite set of *clock variables*. The set $\Phi(C)$ of *clock constraints* is defined by the following BNF notation.

$$\Phi(C) \ni \delta = x < c \mid x > c \mid x \leq c \mid x \geq c \mid \mathbf{true} \mid \delta \wedge \delta \quad \text{where } x \in C \text{ and } c \in \mathbb{Z}_{\geq 0}.$$

Absence of \vee or \neg does not harm expressivity: \vee can be emulated with nondeterminism (see Def. 2.3); and \neg can be propagated down to atomic formulas by the de Morgan

laws. Restriction to **true** and \wedge is technically useful, too, when we deal with intervals and zones (Def. 4.1).

A *clock interpretation* ν over the set C of clock variables is a function $\nu : C \rightarrow \mathbb{R}_{\geq 0}$. Given a clock interpretation ν and $t \in \mathbb{R}_{\geq 0}$, $\nu + t$ denotes the clock interpretation that maps a clock variable $x \in C$ to $\nu(x) + t$.

Definition 2.3 (timed automaton) A *timed automaton* (TA) \mathcal{A} is a tuple $(\Sigma, S, S_0, C, E, F)$ where: Σ is a finite alphabet; S is a finite set of states; $S_0 \subseteq S$ is the set of initial states; C is the set of clock variables; $E \subseteq S \times S \times \Sigma \times \mathcal{P}(C) \times \Phi(C)$ is the set of transitions; and $F \subseteq S$ is the set of accepting states.

The intuition for $(s, s', a, \lambda, \delta) \in E$ is: from s , also assuming that the clock constraint δ is satisfied, we can move to the state s' conducting the action a and resetting the value of each clock variable $x \in \lambda$ to 0. Examples of TAs are in (7) and Fig. 7–11 later.

The above notations (as well as the ones below) follow those in [1]. In the following definition (1) of run, for example, the first transition occurs at (absolute) time τ_1 and the second occurs at time τ_2 ; it is implicit that we stay at the state s_1 for time $\tau_2 - \tau_1$.

Definition 2.4 (run) A *run* of a timed automaton $\mathcal{A} = (\Sigma, S, S_0, C, E, F)$ over a timed word $(\bar{a}, \bar{\tau}) \in (\Sigma \times \mathbb{R}_{>0})^*$ is a pair $(\bar{s}, \bar{\nu}) \in S^* \times ((\mathbb{R}_{\geq 0})^C)^*$ of a sequence \bar{s} of states and a sequence $\bar{\nu}$ of clock interpretations, subject to the following conditions: 1) $|\bar{s}| = |\bar{\nu}| = |\bar{a}| + 1$; 2) $s_0 \in S_0$, and for any $x \in C$, $\nu_0(x) = 0$; and 3) for any $i \in [0, |\bar{a}| - 1]$ there exists a transition $(s_i, s_{i+1}, a_{i+1}, \lambda, \delta) \in E$ such that the clock constraint δ holds under the clock interpretation $\nu_i + (\tau_{i+1} - \tau_i)$ (here τ_0 is defined to be 0), and the clock interpretation ν_{i+1} has it that $\nu_{i+1}(x) = \nu_i(x) + \tau_{i+1} - \tau_i$ (if $x \notin \lambda$) and $\nu_{i+1}(x) = 0$ (if $x \in \lambda$). This run is depicted as follows.

$$(s_0, \nu_0) \xrightarrow{(a_1, \tau_1)} (s_1, \nu_1) \xrightarrow{(a_2, \tau_2)} \dots \longrightarrow (s_{|\bar{a}|-1}, \nu_{|\bar{\tau}|-1}) \xrightarrow{(a_{|\bar{a}|}, \tau_{|\bar{\tau}|})} (s_{|\bar{a}|}, \nu_{|\bar{\tau}|}) \quad (1)$$

Such a run $(\bar{s}, \bar{\nu})$ of \mathcal{A} is *accepting* if $s_{|\bar{s}|-1} \in F$. The *language* $L(\mathcal{A})$ of \mathcal{A} is defined by $L(\mathcal{A}) = \{w \mid \text{there is an accepting run of } \mathcal{A} \text{ over } w\}$.

There is another specification formalism for timed languages called *timed regular expressions* (TREs) [2, 3]. Unlike in the classic Kleene theorem, in the timed case timed automata are strictly more expressive than TREs. See [14, Prop. 2].

Region automaton is an important theoretical gadget in the theory of timed automata: it reduces the domain $S \times (\mathbb{R}_{\geq 0})^C$ of pairs (s, ν) in (1)—that is an *infinite* set—to its *finite* abstraction, the latter being amenable to algorithmic treatments. Specifically it relies on an equivalence relation \sim over clock interpretations. Given a timed automaton $\mathcal{A} = (\Sigma, S, S_0, C, E, F)$ —where, without loss of generality, we assume that each clock variable $x \in C$ appears in at least one clock constraint in E —let c_x denote the greatest number that is compared with x in the clock constraints in E . (Precisely: $c_x = \max\{c \in \mathbb{Z}_{\geq 0} \mid x \bowtie c \text{ occurs in } E, \text{ where } \bowtie \in \{<, >, \leq, \geq\}\}$.) Writing $\text{int}(\tau)$ and $\text{frac}(\tau)$ for the integer and fractional parts of $\tau \in \mathbb{R}_{\geq 0}$, an equivalence relation \sim over clock interpretations ν, ν' is defined as follows. We have $\nu \sim \nu'$ if:

- for each $x \in C$ we have $\text{int}(\nu(x)) = \text{int}(\nu'(x))$ or $(\nu(x) > c_x \text{ and } \nu'(x) > c_x)$;

- for any $x, y \in C$ such that $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $\text{frac}(\nu(x)) < \text{frac}(\nu(y))$ if and only if $\text{frac}(\nu'(x)) < \text{frac}(\nu'(y))$; and
- for any $x \in C$ such that $\nu(x) \leq c_x$, $\text{frac}(\nu(x)) = 0$ if and only if $\text{frac}(\nu'(x)) = 0$.

A *clock region* is an equivalence class of clock interpretations modulo \sim ; as usual the equivalence class of ν is denoted by $[\nu]$. Let α, α' be clock regions. We say α' is a *time-successor* of α if for any $\nu \in \alpha$, there exists $t \in \mathbb{R}_{>0}$ such that $\nu + t \in \alpha'$.

Definition 2.5 (region automaton) For a timed automaton $\mathcal{A} = (\Sigma, S, S_0, C, E, F)$, the *region automaton* $R(\mathcal{A})$ is the NFA $(\Sigma, S', S'_0, E', F')$ defined as follows: $S' = S \times ((\mathbb{R}_{\geq 0})^C / \sim)$; on initial states $S'_0 = \{(s, [\nu]) \mid s \in S_0, \nu(x) = 0 \text{ for each } x \in C\}$; on accepting states $F' = \{(s, \alpha) \in S' \mid s \in F\}$. The transition relation $E' \subseteq S' \times S' \times \Sigma$ is defined as follows: $((s, \alpha), (s', \alpha'), a) \in E'$ if there exist a clock region α'' and $(s, s', a, \lambda, \delta) \in E$ such that

- α'' is a time-successor of α , and
- for each $\nu \in \alpha''$, 1) ν satisfies δ , and 2) there exists $\nu' \in \alpha'$ such that $\nu(x) = \nu'(x)$ (if $x \notin \lambda$) and $\nu'(x) = 0$ (if $x \in \lambda$).

It is known [1] that the region automaton $R(\mathcal{A})$ indeed has finitely many states.

The following notation for NFAs will be used later.

Definition 2.6 ($Runs_{\mathcal{A}}(s, s')$) Let \mathcal{A} be an NFA over Σ , and s and s' be its states. We let $Runs_{\mathcal{A}}(s, s')$ denote the set of runs from s to s' , that is, $Runs_{\mathcal{A}}(s, s') = \{s_0 s_1 \dots s_n \mid n \in \mathbb{Z}_{\geq 0}, s_0 = s, s_n = s', \forall i. \exists a_{i+1}. s_i \xrightarrow{a_{i+1}} s_{i+1} \text{ in } \mathcal{A}\}$.

2.2 String Matching and the Boyer-Moore Algorithm

In §2.2–2.3 we shall revisit the Boyer-Moore algorithm and its adaptation for pattern matching [25]. We do so in considerable details, so as to provide both technical and intuitional bases for our timed adaptation.

String matching is a fundamental operation on strings: given an input string str and a pattern string pat , it asks for the *match set*

$\{(i, j) \mid str(i, j) = pat\}$. An example (from [19]) is in Fig. 1, where the answer is $\{(18, 24)\}$.

A brute-force algorithm has the complexity $O(|str||pat|)$; known optimizations include ones by Knuth, Morris, and Pratt [17] and by Boyer and Moore [7]. The former performs better in the worst case, but for practical instances the latter is commonly used. Let us now demonstrate how the Boyer-Moore algorithm for string matching works, using the example in Fig. 1. Its main idea is to skip unnecessary matching of characters, using two *skip value functions* Δ_1 and Δ_2 (that we define later).

The bottom line in the Boyer-Moore algorithm is that the pattern string pat moves from left to right, and matching between the input string str and pat is conducted from right to left. In (2) is the initial configuration, and we set out with comparing the characters $str(7)$ and $pat(7)$. They turn out to be different.

$str =$	H	E	R	E	I	S	A	S	I	M	P	L	E	E	X	A	M	P	L	E
$pat =$														E	X	A	M	P	L	E
														1	2	3	4	5	6	7

Fig. 1. The string matching problem

A naive algorithm would then move the pattern to the right by one position. We can do better, however, realizing that the character $str(7) = S$ (that we already read for comparison) never occurs in the pattern pat . This means the

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
H E R E I S A S I M P L E E X A M P L E
E X A M P L E (2)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
H E R E I S A S I M P L E E X A M P L E
E X A M P L E (3)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
H E R E I S A S I M P L E E X A M P L E
E X A M P L E (4)

position 7 cannot belong to any matching interval (i, j) , and we thus jump to the configuration (3). Formally this argument is expressed by the value $\Delta_1(S, 7) = 7$ of the first skip value function Δ_1 , as we will see later.

Here again we compare characters from right to left, in (3), realizing immediately that $str(14) \neq pat(7)$. It is time to shift the pattern; given that $str(14) = P$ occurs as $pat(5)$, we shift the pattern by $\Delta_1(P, 7) = 7 - 5 = 2$.

We are now in the configuration (4), and some initial matching succeeds ($str(16) = pat(7)$, $str(15) = pat(6)$, and so on). The matching fails for $str(12) \neq pat(3)$. Following the same reasoning as above—the character $str(12) = I$ does not occur in $pat(3)$, $pat(2)$ or $pat(1)$ —we would then shift the pattern by $\Delta_1(I, 3) = 3$.

However we can do even better. Consider the table on the right, where we forget about the input str and shift the pattern pat one by one, trying to match it with pat itself. We are specifically interested in the segment MPLE from $pat(4)$ to $pat(7)$ (underlined in the first row)—because it is the partial match we have discovered in the configuration (4). The table shows that we need to

	1	2	3	4	5	6	7
	E	X	A	M	P	L	E
✗	*	E	X	A	M	P	L
✗	*	*	E	X	A	M	P
⋮							
⋮							
✗	*	*	*	*	*	E	X
✓	*	*	*	*	*	E	X

Fig. 2. Table for computing Δ_2

shift at least by 6 to get a potential match (the last row);

hence from the configuration (4) we can shift the pattern pat by 6, which is more than the skip value in the above ($\Delta_1(I, 3) = 3$). This argument—different from the one for Δ_1 —is formalized as the second skip value function $\Delta_2(3) = 6$.

We are led to the configuration on the right, only to find that the first matching trial fails ($str(22) \neq pat(7)$).

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
H E R E I S A S I M P L E E X A M P L E
E X A M P L E (5)

Since $str(22) = P$ occurs in pat as $pat(5)$, we shift pat by $\Delta_1(P, 7) = 2$. This finally brings us to the configuration in Fig. 1 and the match set $\{(18, 24)\}$.

Summarizing, the key in the Boyer-Moore algorithm is to use two skip value functions Δ_1, Δ_2 to shift the pattern faster than one-by-one. The precise definition of Δ_1, Δ_2 is in Appendix A, for reference.

2.3 Pattern Matching and a Boyer-Moore type Algorithm

Pattern matching is another fundamental operation that generalizes string matching: given an input string str and a regular language L as a *pattern*, it asks for the *match set* $\{(i, j) \mid str(i, j) \in L\}$. For example, for str in Fig. 1 and the pattern $[A-Z]^*MPLE$, the match set is $\{(11, 16), (18, 24)\}$. In [25] an algorithm for pattern matching is introduced that employs “Boyer-Moore type” optimization, much like the use of Δ_2 in §2.2.

Let $str = cbadcdc$ be an input string and $dc^*\{ba \mid dc\}$ be a pattern L , for example. We can solve pattern matching by the following brute-force algorithm.

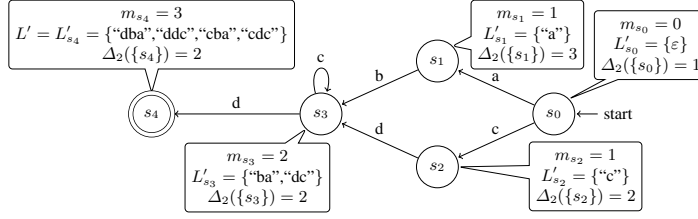


Fig. 3. The automaton \mathcal{A} and skip values

- We express the pattern as an NFA $\mathcal{A} = (\Sigma, S, S_0, E, F)$ in Fig. 3. We reverse words— $w \in L$ if and only if $w^{Rev} \in L(\mathcal{A})$ —following the Boyer-Moore algorithm (§2.2) that matches a segment of input and a pattern *from right to left*.
- Also following §2.2 we “shift the pattern from left to right.” This technically means: we conduct the following for $j = 1$ first, then $j = 2$, and so on. For fixed j we search for $i \in [1, j]$ such that $str(i, j) \in L$. This is done by computing the set $S_{(i,j)}$ of reachable states of \mathcal{A} when it is fed with $str(i, j)^{Rev}$. The computation is done step-by-step, decrementing i from j to 1:

$$S_{(j,j)} = \{s \mid \exists s' \in S_0. s' \xrightarrow{str(j)} s\}, \text{ and } S_{(i,j)} = \{s \mid \exists s' \in S_{(i+1,j)}. s' \xrightarrow{str(i)} s\}.$$

Then (i, j) is a matching interval if and only if $S_{(i,j)} \cap F \neq \emptyset$. See Fig. 4.

The Boyer-Moore type optimization in [25] tries to hasten the shift of j . A key observation is as follows. Assume $j = 4$; then the above procedure would feed $str(1, 4)^{Rev} = dabc$ to the automaton \mathcal{A} (Fig. 3). We instantly see that this would not yield any matching interval—for a word to be accepted by \mathcal{A} it must start with abd , cdd , abc or cdc .

Precisely the algorithm in [25] works as follows. We first observe that the shortest word accepted by \mathcal{A} is 3; therefore we can start right away with $j = 3$, skipping $j = 1, 2$ in the above brute-force algorithm. Unfortunately $str(1, 3) = cba$ does not match L , with $str(1, 3)^{Rev} = abc$ only leading to $\{s_3\}$ in \mathcal{A} .

We now shift j by 2, from 3 to 5, following Fig. 5. Here

$$L' = \{ (w(1, 3))^{Rev} \mid w \in L(\mathcal{A}) \} = \{dba, ddc, cba, cdc\}; \quad (6)$$

that is, for $str(i, j)$ to match the pattern L its last three characters must match L' . Our previous “key observation” now translates to the fact that $str(2, 4) = bad$ does not belong to L' ; in the actual algorithm in [25], however, we do not use the string $str(2, 4)$ itself. Instead we *overapproximate* it with the information that feeding \mathcal{A} with $str(1, 3)^{Rev} = abc$ led to $\{s_3\}$. Similarly to the case with L' , this implies that the last two characters of $str(1, 3)$ must have been in $L'_{s_3} = \{ba, dc\}$. The table shows that none in L'_{s_3} matches any of L' when j is shifted by 1; when j is shifted by 2, we have matches (underlined). Therefore we jump from $j = 3$ to $j = 5$.

This is how the algorithm in [25] works: it accelerates the brute-force algorithm in Fig. 4, skipping some j 's, with the help of a skip value function Δ_2 . The overapproximation in the last paragraph allows Δ_2 to rely only on a pattern L (but not on an input string str); this means that pre-processing is done once we fix the pattern L , and it is reused for various input strings str . This is an advantage in monitoring applications

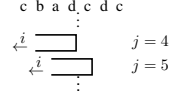


Fig. 4. A brute-force algorithm

	2	3	4	5
L'_{s_3}	b	a		
	d	c		
L'	d	b	a	
shifted by 1	d	d	c	
	c	b	a	
(X)	c	d	c	
L'	*	d	b	a
shifted by 2	*	d	d	c
(✓)	<u>c</u>	<u>b</u>	a	
	<u>c</u>	<u>d</u>	c	

Fig. 5. Table for $\Delta_2(s_3)$

where one would deal with a number of input strings str , some of which are yet to come. See Appendix B for the precise definition of the skip value function Δ_2 .

In Fig. 3 we annotate each state s with the values m_s and L'_s that is used in computing $\Delta_2(\{s\})$. Here m_s is the length of a shortest word that leads to s ; $m = \min_{s \in F} m_s$ (that is 3 in the above example); and $L'_s = \{w(1, \min\{m_s, m\})^{Rev} \mid w \in L(\mathcal{A}_s)\}$.

It is not hard to generalize the other skip value function Δ_1 in §2.2 for pattern matching, too: in place of pat we use the set L' in the above (6). See Appendix B.

3 The Timed Pattern Matching Problem

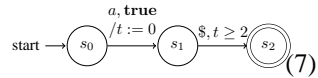
Here we formulate our problem, following the notations in §2.1.

Given a timed word w , the timed word segment $w|_{(t,t')}$ is the result of clipping the parts outside the open interval (t, t') . For example, for $w = ((a, b, c), (0.7, 1.2, 1.5))$, we have $w|_{(1.0, 1.7)} = ((b, c, \$), (0.2, 0.5, 0.7))$, $w|_{(1.0, 1.5)} = ((b, \$), (0.2, 0.5))$ and $w|_{(1.2, 1.5)} = ((\$), (0.3))$. Here the (fresh) *terminal character* $\$$ designates the end of a segment. Since we use open intervals (t, t') , for example, the word $w|_{(1.2, 1.5)}$ does not contain the character c at time 0.3. The formal definition is as follows.

Definition 3.1 (timed word segment) Let $w = (\bar{a}, \bar{\tau})$ be a timed word over Σ , t and t' be reals such that $0 \leq t < t'$, and i and j be indices such that $\tau_{i-1} \leq t < \tau_i$ and $\tau_j < t' \leq \tau_{j+1}$ (we let $\tau_0 = 0$ and $\tau_{|\bar{\tau}|+1} = \infty$). The *timed word segment* $w|_{(t,t')}$ of w on the interval (t, t') is the timed word $(\bar{a}', \bar{\tau}')$, over the extended alphabet $\Sigma \amalg \{\$\}$, defined as follows: 1) $|w|_{(t,t')}| = j - i + 2$; 2) we have $a'_k = a_{i+k-1}$ and $\tau'_k = \tau_{i+k-1} - t$ for $k \in [1, j - i + 1]$; and 3) $a'_{j-i+2} = \$$ and $\tau'_{j-i+2} = t' - t$.

Definition 3.2 (timed pattern matching) The *timed pattern matching* problem (over an alphabet Σ) takes (as input) a timed word w over Σ and a timed automaton \mathcal{A} over $\Sigma \amalg \{\$\}$; and it requires (as output) the *match set* $\mathcal{M}(w, \mathcal{A}) = \{(t, t') \in (\mathbb{R}_{\geq 0})^2 \mid t < t', w|_{(t,t')} \in L(\mathcal{A})\}$.

Our formulation in Def. 3.2 slightly differs from that in [22] in that: 1) we use timed words in place of signals (Rem. 2.2); 2) for specification we use timed automata rather than timed regular expressions; and 3) we use an explicit terminal character $\$$.



While none of these differences is major, introduction of $\$$ enhances expressivity, e.g. in specifying “an event a occurs, and after that, no other event occurs within 2s.” (see (7)). It is also easy to ignore $\$$ —when one is not interested in it—by having the clock constraint **true** on the $\$$ -labeled transitions leading to the accepting states. **Assumption 3.3** In what follows we assume the following. Each timed automaton \mathcal{A} over the alphabet $\Sigma \amalg \{\$\}$ is such that: every $\$$ -labeled transition is into an accepting state; and no other transition is $\$$ -labeled. And there exists no transition from any accepting states.

4 A Naive Algorithm and Its Online Variant

Here we present a naive algorithm for timed pattern matching (without a Boyer-Moore type optimization), also indicating how to make it into an online one. Let us fix a timed word w over Σ and a timed automaton $\mathcal{A} = (\Sigma \amalg \{\$\}, S, S_0, C, E, F)$ as the input.

First of all, a match set (Def. 3.2) is in general an infinite set, and we need its finitary representation for an algorithmic treatment. We follow [22] and use (2-dimensional) zones for that purpose.

Definition 4.1 (zone) Consider the 2-dimensional plane \mathbb{R}^2 whose axes are denoted by t and t' . A *zone* is a convex polyhedron specified by constraints of the form $t \bowtie c$, $t' \bowtie c$ and $t' - t \bowtie c$, where $\bowtie \in \{<, >, \leq, \geq\}$ and $c \in \mathbb{Z}_{\geq 0}$.

It is not hard to see that each zone is specified by three intervals (that may or may not include their endpoints): T_0 for t , T_f for t' and T_Δ for $t' - t$. We let a triple (T_0, T_f, T_Δ) represent a zone, and write $(t, t') \in (T_0, T_f, T_\Delta)$ if $t \in T_0$, $t' \in T_f$ and $t' - t \in T_\Delta$.

In our algorithms we shall use the following constructs.

Definition 4.2 (reset, eval, solConstr, ρ_\emptyset , Conf) Let $\rho: C \rightarrow \mathbb{R}_{>0}$ be a partial function that carries a clock variable $x \in C$ to a positive real; the intention is that x was reset at time $\rho(x)$ (in the absolute clock). Let $x \in C$ and $t_r \in \mathbb{R}_{>0}$; then the partial function $\text{reset}(\rho, x, t_r): C \rightarrow \mathbb{R}_{>0}$ is defined by: $\text{reset}(\rho, x, t_r)(x) = t_r$ and $\text{reset}(\rho, x, t_r)(y) = \rho(y)$ for each $y \in C$ such that $y \neq x$. (The last is Kleene's equality between partial functions, to be precise.)

Now let ρ be as above, and $t, t_0 \in \mathbb{R}_{\geq 0}$, with the intention that t is the current (absolute) time and t_0 is the epoch (absolute) time for a timed word segment $w|_{(t_0, t']}$. We further assume $t_0 \leq t$ and $t_0 \leq \rho(x) \leq t$ for each $x \in C$ for which $\rho(x)$ is defined. The clock interpretation $\text{eval}(\rho, t, t_0): C \rightarrow \mathbb{R}_{\geq 0}$ is defined by: $\text{eval}(\rho, t, t_0)(x) = t - \rho(x)$ (if $\rho(x)$ is defined); and $\text{eval}(\rho, t, t_0)(x) = t - t_0$ (if $\rho(x)$ is undefined).

For intervals $T, T' \subseteq \mathbb{R}_{\geq 0}$, a partial function $\rho: C \rightarrow \mathbb{R}_{>0}$ and a clock constraint δ (§2.1), we define $\text{solConstr}(T, T', \rho, \delta) = \{(t, t') \mid t \in T, t' \in T', \text{eval}(\rho, t', t) \models \delta\}$. We let $\rho_\emptyset: C \rightarrow \mathbb{R}_{>0}$ denote the partial function that is nowhere defined.

For a timed word w , a timed automaton \mathcal{A} and each $i, j \in [1, |w|]$, we define the set of “configurations”: $\text{Conf}(i, j) = \{(s, \rho, T) \mid \forall t_0 \in T. \exists (\bar{s}, \bar{\nu}). (\bar{s}, \bar{\nu}) \text{ is a run over } w(i, j) - t_0, s|_{\bar{s}-1} = s, \text{ and } \nu|_{\bar{\nu}-1} = \text{eval}(\rho, \tau_j, t_0)\}$. Further details are in Appendix C.

Our first (naive) algorithm for timed pattern matching is in Alg. 1. We conduct a brute-force breadth-first search, computing $\{(t, t') \in \mathcal{M}(w, \mathcal{A}) \mid \tau_{i-1} \leq t < \tau_i, \tau_j < t' \leq \tau_{j+1}\}$ for each i, j , with the aid of $\text{Conf}(i, j)$ in Def. 4.2. (The singular case of $\forall i. \tau_i \notin (t, t')$ is separately taken care of by *Immd.*) We do so in the order illustrated in Fig. 6: we decrement i , and for each i we increment j . This order—that flips the one in Fig. 4—is for the purpose of the Boyer-Moore type optimization later in §5. In Appendix C we provide further details.

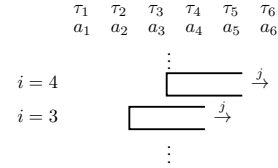


Fig. 6. i, j in our algorithms for timed pattern matching

Theorem 4.3 (termination and correctness of Alg. 1) 1. Alg. 1 terminates and its answer Z is a finite union of zones.

2. For any $t, t' \in \mathbb{R}_{>0}$ such that $t < t'$, the following are equivalent: 1) there is a zone $(T_0, T_f, T_\Delta) \in Z$ such that $(t, t') \in (T_0, T_f, T_\Delta)$; and 2) there is an accepting run $(\bar{s}, \bar{\nu})$ over $w|_{(t, t']}$ of \mathcal{A} . \square

As an immediate corollary, we conclude that a match set $\mathcal{M}(w, \mathcal{A})$ always allows representation by finitely many zones.

Changing the order of examination of i, j (Fig. 6) gives us an *online* variant of Alg. 1. It is presented in Appendix D; nevertheless our Boyer-Moore type algorithm is based on the original Alg. 1.

5 A Timed Boyer-Moore Type Algorithm

Here we describe our main contribution, namely a Boyer-Moore type algorithm for timed pattern matching. Much like the algorithm in [25] skips some j 's in Fig. 4 (§2.3), we wish to skip some i 's in Fig. 6. Let us henceforth fix a timed word $w = (\bar{a}, \bar{\tau})$ and a timed automaton $\mathcal{A} = (\Sigma \amalg \{\$, S, S_0, C, E, F)$ as the input of the problem.

Let us define the *optimal* skip value function by $Opt(i) = \min\{n \in \mathbb{R}_{>0} \mid \exists t \in [\tau_{i-n-1}, \tau_{i-n}). \exists t' \in (t, \infty). (t, t') \in \mathcal{M}(w, \mathcal{A})\}$; the value $Opt(i)$ designates the biggest skip value, at each i in Fig. 6, that does not change the outcome of the algorithm. Since the function Opt is not amenable to efficient computation in general, our goal is its underapproximation that is easily computed.

Towards that goal we follow the (untimed) pattern matching algorithm in [25]; see §2.3. In applying the same idea as in Fig. 5 to define a skip value, however, the first obstacle is that the language L'_{s_3} —the set of (suitable prefixes of) all words that lead to s_3 —becomes an *infinite* set in the current timed setting. Our countermeasure is to use a *region automaton* $R(\mathcal{A})$ (Def. 2.5) for representing the set.

We shall first introduce some constructs used in our algorithm.

Definition 5.1 ($\mathcal{W}(r), \mathcal{W}(\bar{s}, \bar{\alpha})$) Let r be a set of runs of the timed automaton \mathcal{A} . We define a timed language $\mathcal{W}(r)$ by: $\mathcal{W}(r) = \{(\bar{a}, \bar{\tau}) \mid \text{in } r \text{ there is a run of } \mathcal{A} \text{ over } (\bar{a}, \bar{\tau})\}$.

For the region automaton $R(\mathcal{A})$, each run $(\bar{s}, \bar{\alpha})$ of $R(\mathcal{A})$ —where $s_k \in S$ and $\alpha_k \in (\mathbb{R}_{\geq 0})^C / \sim$, recalling the state space of $R(\mathcal{A})$ from Def. 2.5—is naturally identified with a set of runs of \mathcal{A} , namely $\{(\bar{s}, \bar{\nu}) \in (S \times (\mathbb{R}_{\geq 0})^C)^* \mid \nu_k \in \alpha_k \text{ for each } k\}$. Under this identification we shall sometimes write $\mathcal{W}(\bar{s}, \bar{\alpha})$ for a suitable timed language, too.

The above definitions of $\mathcal{W}(r)$ and $\mathcal{W}(\bar{s}, \bar{\alpha})$ naturally extends to a set r of *partial runs* of \mathcal{A} , and to a *partial run* $(\bar{s}, \bar{\alpha})$ of $R(\mathcal{A})$, respectively. Here a partial run is a run but we do not require: it start at an initial state; or its initial clock interpretation be 0. The next optimization of $R(\mathcal{A})$ is similar to so-called *trimming*, but we leave those states that do not lead to any final state (they become necessary later).

Definition 5.2 ($R^r(\mathcal{A})$) For a timed automaton \mathcal{A} , we let $R^r(\mathcal{A})$ denote its *reachable region automaton*. It is the NFA $R^r(\mathcal{A}) = (\Sigma, S^r, S_0^r, E^r, F^r)$ obtained from $R(\mathcal{A})$ (Def. 2.5) by removing all those states which are unreachable from any initial state.

We are ready to describe our Boyer-Moore type algorithm. We use a skip value function Δ_2 that is similar to the one in §2.3 (see Fig. 3 & 5), computed with the aid of m_s and L'_s defined for each state s . We define m_s and L'_s using the NFA $R^r(\mathcal{A})$. Notable differences are: 1) here L'_s and L' are sets of *runs*, not of *words*; and 2) since the orders are flipped between Fig. 4 & 6, *Rev* e.g. in (6) is gone now.

The precise definitions are as follows. Here $s \in S$ is a state of the (original) timed automaton \mathcal{A} ; and we let $R^r(s) = \{(s, \alpha) \in S^r\}$.

$$\begin{aligned} m &= \min\{|w'| \mid w' \in L(\mathcal{A})\} & m_s &= \min\{|r| \mid \beta_0 \in S_0^r, \beta \in R^r(s), r \in \text{Runs}_{R^r(\mathcal{A})}(\beta_0, \beta)\} \\ L' &= \{r(0, m-1) \mid \beta_0 \in S_0^r, \beta_f \in F^r, r \in \text{Runs}_{R^r(\mathcal{A})}(\beta_0, \beta_f)\} \\ L'_s &= \{r(0, \min\{m, m_s\}-1) \mid \beta_0 \in S_0^r, \beta \in R^r(s), r \in \text{Runs}_{R^r(\mathcal{A})}(\beta_0, \beta)\} \end{aligned} \quad (8)$$

Algorithm 1 Our naive algorithm for timed pattern matching. See Def. 4.2 and Appendix C for details.

Require: A timed word $w = (\bar{a}, \bar{\tau})$, and a timed automaton $\mathcal{A} = (\Sigma, S, S_0, C, E, F)$.
Ensure: $\bigcup Z$ is the match set $\mathcal{M}(w, \mathcal{A})$ in Def. 3.2.

```

1:  $i \leftarrow |w|$ ;  $CurrConf \leftarrow \emptyset$ ;  $Immd \leftarrow \emptyset$ ;  $Z \leftarrow \emptyset$   $\triangleright$   $Immd$  and  $Z$  are finite sets of zones.
2: for  $s \in S_0$  do  $\triangleright$  Lines 2–5 compute  $Immd$ .
3:   for  $s_f \in F$  do
4:     for  $(s, s_f, \$, \lambda, \delta) \in E$  do
5:        $Immd \leftarrow Immd \cup \text{solConstr}([0, \infty), (0, \infty), \rho_\emptyset, \delta)$ 
6:  $Z \leftarrow Z \cup \{ (T_0 \cap [\tau_{|w|}, \infty), T_f \cap (\tau_{|w|}, \infty), T_\Delta) \mid (T_0, T_f, T_\Delta) \in Immd \}$ 
7:  $\triangleright$  We have added  $\{ (t, t') \in \mathcal{M}(w, \mathcal{A}) \mid t, t' \in [\tau_{|w|}, \infty) \}$  to  $Z$ .
8: while  $i > 0$  do
9:    $Z \leftarrow Z \cup \{ (T_0 \cap [\tau_{i-1}, \tau_i], T_f \cap (\tau_{i-1}, \tau_i], T_\Delta) \mid (T_0, T_f, T_\Delta) \in Immd \}$ 
10:   $\triangleright$  We have added  $\{ (t, t') \in \mathcal{M}(w, \mathcal{A}) \mid t, t' \in [\tau_{i-1}, \tau_i] \}$  to  $Z$ .
11:   $\triangleright$  Now, for each  $j$ , we shall add  $\{ (t, t') \in \mathcal{M}(w, \mathcal{A}) \mid t \in [\tau_{i-1}, \tau_i], t' \in (\tau_j, \tau_{j+1}] \}$ .
12:   $j \leftarrow i$ 
13:   $CurrConf \leftarrow \{ (s, \rho_\emptyset, [\tau_{i-1}, \tau_i]) \mid s \in S_0 \}$ 
14:  while  $CurrConf \neq \emptyset$  &  $j \leq |w|$  do
15:     $(PrevConf, CurrConf) \leftarrow (CurrConf, \emptyset)$   $\triangleright$  Here  $PrevConf = Conf(i, j-1)$ .
16:    for  $(s, \rho, T) \in PrevConf$  do
17:      for  $(s, s', a_j, \lambda, \delta) \in E$  do  $\triangleright$  Read  $(a_j, \tau_j)$ .
18:         $T' \leftarrow \{ t_0 \in T \mid \text{eval}(\rho, \tau_j, t_0) \models \delta \}$ 
19:         $\triangleright$  Narrow the interval  $T$  to satisfy the clock constraint  $\delta$ .
20:        if  $T' \neq \emptyset$  then
21:           $\rho' \leftarrow \rho$ 
22:          for  $x \in \lambda$  do
23:             $\rho' \leftarrow \text{reset}(\rho', x, \tau_j)$   $\triangleright$  Reset the clock variables in  $\lambda$ .
24:           $CurrConf \leftarrow CurrConf \cup (s', \rho', T')$ 
25:          for  $s_f \in F$  do  $\triangleright$  Lines 25–31 try to insert  $\$$  in  $(\tau_j, \tau_{j+1}]$ .
26:            for  $(s', s_f, \$, \lambda', \delta') \in E$  do
27:              if  $j = |w|$  then
28:                 $T'' \leftarrow (\tau_j, \infty)$ 
29:              else
30:                 $T'' \leftarrow (\tau_j, \tau_{j+1}]$ 
31:                 $Z \leftarrow Z \cup \text{solConstr}(T', T'', \rho', \delta')$ 
32:             $j \leftarrow j + 1$ 
33:           $i \leftarrow i - 1$ 

```

Note again that these data are defined over $R^r(\mathcal{A})$ (Def. 5.2); $Runs_{R^r(\mathcal{A})}(\beta_0, \beta_f)$ is from Def. 2.6.

Definition 5.3 (Δ_2) Let $Conf$ be a set of triples (s, ρ, T) of: a state $s \in S$ of \mathcal{A} , $\rho: C \rightarrow \mathbb{R}_{>0}$, and an interval T . (This is much like $Conf(i, j)$ in Def. 4.2.) We define the skip value $\Delta_2(Conf)$ as follows.

$$\begin{aligned}
d_1(r) &= \min_{r' \in L'} \min\{n \in \mathbb{Z}_{>0} \mid \mathcal{W}(r) \cap \left(\bigcup_{r'' \in \text{pref}(r'(n, |r'|))} \mathcal{W}(r'') \right) \neq \emptyset\} \\
d_2(r) &= \min_{r' \in L'} \min\{n \in \mathbb{Z}_{>0} \mid \left(\bigcup_{r'' \in \text{pref}(r)} \mathcal{W}(r'') \right) \cap \mathcal{W}(r'(n, |r'|)) \neq \emptyset\} \\
\Delta_2(Conf) &= \max_{(s, \rho, T) \in Conf} \min_{r \in L'_s} \min\{d_1(r), d_2(r)\}
\end{aligned}$$

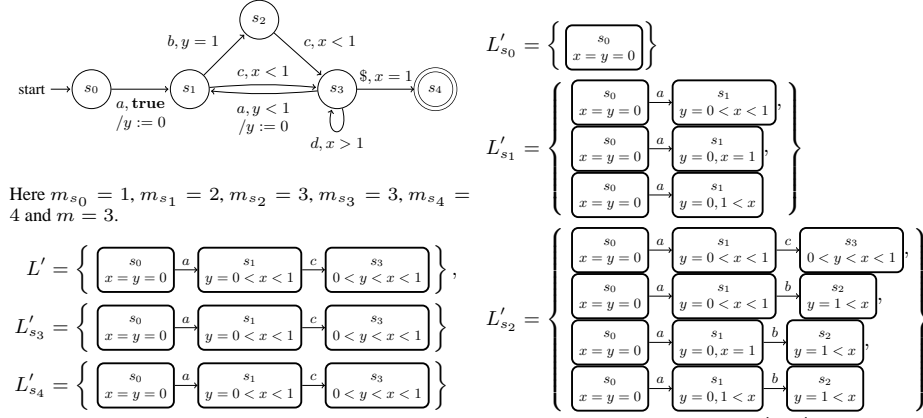


Fig. 7. An example of a timed automaton \mathcal{A} , and the values m_s, L'_s, L'

Here $r \in \text{Runs}_{R^r}(\mathcal{A})$; L' is from (8); \mathcal{W} is from Def. 5.1; $r'(n, |r'|)$ is a subsequence of r' (that is a partial run); and $\text{pref}(r)$ denotes the set of all prefixes of r .

Theorem 5.4 (correctness of Δ_2) *Let $i \in [1, |w|]$, and $j = \max\{j \in [i, |w|] \mid \text{Conf}(i, j) \neq \emptyset\}$, where $\text{Conf}(i, j)$ is from Def. 4.2. (In case $\text{Conf}(i, j)$ is everywhere empty we let $j = i$.) Then we have $\Delta_2(\text{Conf}(i, j)) \leq \text{Opt}(i)$.* \square

The remaining issue in Def. 5.3 is that the sets like $\mathcal{W}(r)$ and $\mathcal{W}(r'')$ can be infinite—we need to avoid their direct computation. We rely on the usual automata-theoretic trick: the intersection of languages is recognized by a product automaton.

Given two timed automata \mathcal{B} and \mathcal{C} , we let $\mathcal{B} \times \mathcal{C}$ denote their *product* defined in the standard way (see e.g. [21]). The following is straightforward.

Proposition 5.5 *Let $r = (\overline{s}, \overline{\alpha})$ and $r' = (\overline{s'}, \overline{\alpha'})$ be partial runs of $R(\mathcal{B})$ and $R(\mathcal{C})$, respectively; they are naturally identified with sets of partial runs of \mathcal{B} and \mathcal{C} (Def. 5.1). Assume further that $|r| = |r'|$. Then we have $\mathcal{W}(r) \cap \mathcal{W}(r') = \mathcal{W}(r, r')$, where (r, r') is the following set of runs of $\mathcal{B} \times \mathcal{C}$: $(r, r') = \{((\overline{s}, \overline{s'}), (\overline{v}, \overline{v'})) \mid (\overline{s}, \overline{v}) \in (\overline{s}, \overline{\alpha}) \text{ and } (\overline{s'}, \overline{v'}) \in (\overline{s'}, \overline{\alpha'})\}$.* \square

The proposition allows the following algorithm for the emptiness check required in computing d_1 (Def. 5.3). Firstly we distribute \cap over \bigcup ; then we need to check if $\mathcal{W}(r) \cap \mathcal{W}(r'') \neq \emptyset$ for each r'' . The proposition reduces this to checking if $\mathcal{W}(r, r'') \neq \emptyset$, that is, if (r, r'') is a (legitimate) partial run of the region automaton $R(\mathcal{A} \times \mathcal{A})$. The last check is obviously decidable since $R(\mathcal{A} \times \mathcal{A})$ is finite. For d_2 the situation is similar.

We also note that the computation of Δ_2 (Def. 5.3) can be accelerated by memorizing the values $\min_{r \in L'_s} \min\{d_1(r), d_2(r)\}$ for each s .

Finally our Boyer-Moore type algorithm for timed pattern matching is Alg. 3 in Appendix E. Its main differences from the naive one (Alg. 1) are: 1) initially we start with $i = |w| - m + 1$ instead of $i = |w|$ (line 1 of Alg. 1); and 2) we decrement i by the skip value computed by Δ_2 , instead of by 1 (line 33 of Alg. 1).

It is also possible to employ an analog of the skip value function Δ_1 in §2.2 & 2.3. For $c \in \Sigma$ and $p \in \mathbb{Z}_{>0}$, we define $\Delta_1(c, p) = \min_{k > 0} \{k - p \mid k > m \text{ or } \exists (\overline{a}, \overline{\tau}) \in \mathcal{W}(L'). a_k = c\}$. Here m and L' are from (8). Then we can possibly skip more i 's using both Δ_1 and Δ_2 ; see Appendix E for details. In our implementation we do not use Δ_1 ,

though, following the (untimed) pattern matching algorithm in [25]. Investigating the effect of additionally using Δ_1 is future work.

One may think of the following alternative for pattern matching: we first forget about time stamps, time constraints, etc.; the resulting “relaxed” untimed problem can be solved by the algorithm in [25] (§2.3); and then we introduce the time constraints back and refine the interim result to the correct one. Our *timed* Boyer-Moore algorithm has greater skip values in general, however, because by using region automata $R(\mathcal{A})$, $R^r(\mathcal{A})$ we also take time constraints into account when computing skip values.

Remark 5.6 It was suggested by multiple reviewers that our use of region automata be replaced with that of *zone automata* (see e.g. [5]). This can result in a much smaller automaton $R(\mathcal{A})$ for calculating skip values (cf. Def. 5.3 and Case 2 of §6). More importantly, zone automata are insensitive to the time unit size—unlike region automata where the numbers c_x in Def. 2.5 govern their size—a property desired in actual deployment of timed pattern matching. This is a topic of our imminent future work.

6 Experiments

We implemented both of our naive offline algorithm and our Boyer-Moore type algorithm (without Δ_1) in C++ [24]. We ran our experiments on MacBook Air Mid 2011 with Intel Core i7-2677M 1.80GHz CPU with 3.7GB RAM and Arch Linux (64-bit). Our programs were compiled with GCC 5.3.0 with optimization level O3.

An execution of our Boyer-Moore type algorithm consists of two phases: in the first *pre-processing* phase we compute the skip value function Δ_2 —to be precise the value $\min_{r \in L'_s} \min\{d_1(r), d_2(r)\}$ for each s , on which Δ_2 relies—and in the latter “matching” phase we actually compute the match set.

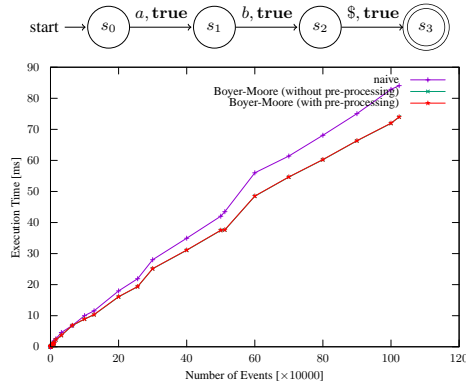
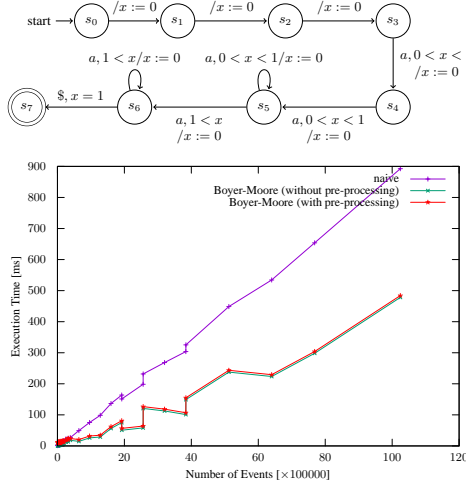
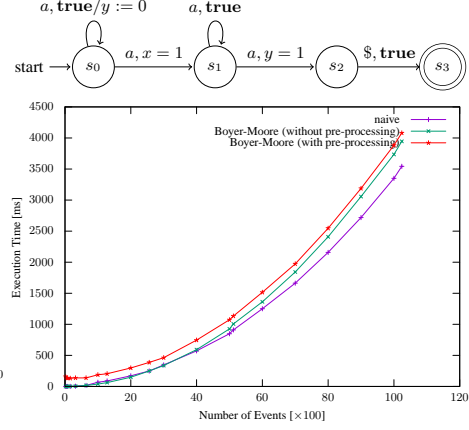
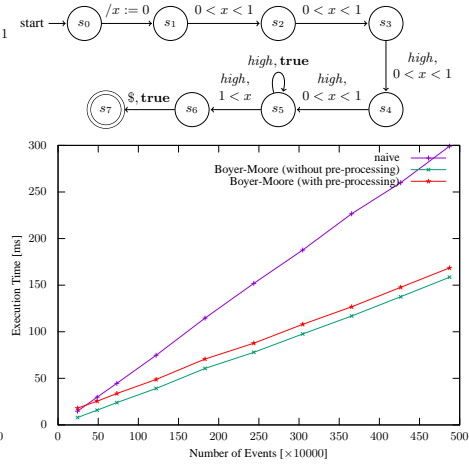
As input we used five test cases: each case consists of a timed automaton \mathcal{A} and multiple timed words w of varying length $|w|$. Cases 1 & 4 are from a previous work [22] on timed pattern matching; in Case 2 the timed automaton \mathcal{A} is not expressible with a timed regular expression (TRE, see [14] and §2.1); and Cases 3 & 5 are our original. In particular Case 5 comes from an automotive example.

Our principal interest is in the relationship between execution time and the length $|w|$ (i.e. the number of events), for both of the naive and Boyer-Moore algorithms. For each test case we ran our programs 30 times; the presented execution time is the average. We measured the execution time separately for the pre-processing phase and the (main) matching phase; in the figures we present the time for the latter.

We present an overview of the results. Further details are found in Appendix G.

Case 1: No Clock Constraints In Fig. 8 we present a timed automaton \mathcal{A} and the execution time (excluding pre-processing) for 37 timed words w whose lengths range from 20 to 1,024,000. Each timed word w is an alternation of $a, b \in \Sigma$, and its time stamps are randomly generated according to a certain uniform distribution.

The automaton \mathcal{A} is without any clock constraints, so in this case the problem is almost that of *untimed* pattern matching. The Boyer-Moore algorithm outperforms the naive one; but the gap is approximately 1/10 when $|w|$ is large enough, which is smaller than what one would expect from the fact that i is always decremented by 2. This is

Fig. 8. Case 1: \mathcal{A} and execution timeFig. 9. Case 3: \mathcal{A} and execution timeFig. 10. Case 2: \mathcal{A} and execution timeFig. 11. Case 5: \mathcal{A} and execution time

because, as some combinatorial investigation would reveal, those i 's which are skipped are for which we examine fewer j 's

The pre-processing phase (that relies only on \mathcal{A}) took $6.38 \cdot 10^{-2}$ ms. on average.

Case 2: Beyond Expressivity of TREs In Fig. 10 is a timed automaton \mathcal{A} —one that is not expressible with TREs [14]—and the execution time for 20 timed words w whose lengths range from 20 to 10,240. Each w is a repetition of $a \in \Sigma$, and its time stamps are randomly generated according to the uniform distribution in the interval $(0, 0.1)$.

One can easily see that the skip value is always 1, so our Boyer-Moore algorithm is slightly slower due to the overhead of repeatedly reading the result of pre-processing. The naive algorithm (and hence the Boyer-Moore one too) exhibits non-linear increase in Fig. 10; this is because its worst-case complexity is bounded by $|w||E|^{|w|+1}$ (where $|E|$ is the number of edges in \mathcal{A}). See the proof of Thm. 4.3 (Appendix F.1). The factor $|E|$ in the above complexity bound stems essentially from nondeterminism.

The pre-processing phase took $1.39 \cdot 10^2$ ms. on average.

Case 3: Accepting Runs Are Long In Fig. 9 are a timed automaton \mathcal{A} and the execution time for 49 timed words w whose lengths range from 8,028 to 10,243,600. Each w is randomly generated as follows: it is a repetition of $a \in \Sigma$; a is repeated according to the exponential distribution with a parameter λ ; and we do so for a fixed duration $\tau_{|\tau|}$, generating a timed word of length $|w|$. See Table 3 in Appendix G.

In the automaton \mathcal{A} the length m of the shortest accepting run is large; hence so are the skip values in the Boyer-Moore optimization. (Specifically the skip value is 5 if both $\tau_i - \tau_{i-1}$ and $\tau_{i+1} - \tau_i$ are greater than 1.) Indeed, as we see from the figure, the Boyer-Moore algorithm outperforms the naive one roughly by twice.

The pre-processing phase took 7.02ms. on average. This is in practice negligible; recall that pre-processing is done only once when \mathcal{A} is given.

Case 4: Region Automata are Big Here \mathcal{A} is a translation of the TRE $\langle (\langle p \rangle_{(0,10]} \langle \neg p \rangle_{(0,10]}^* \wedge (\langle q \rangle_{(0,10]} \langle \neg q \rangle_{(0,10]}^*)^*) \rangle_{(0,80]} \$ \rangle_{(0,80]}$. We executed our two algorithms for 12 timed words w whose lengths range from 1,934 to 31,935. Each w is generated randomly as follows: it is the interleaving combination of an alternation of $p, \neg p$ and one of $q, \neg q$; in each alternation the time stamps are governed by the exponential distribution with a parameter λ ; and its duration $\tau_{|\tau|}$ is fixed. See Table 4 in Appendix G.

This \mathcal{A} is bad for our Boyer-Moore type algorithm since its region automaton $R(\mathcal{A})$ is very big. Specifically: the numbers c_x in Def. 2.5 are big (10 and 80) and we have to have many states accordingly in $R(\mathcal{A})$ —recall that in \sim we care about the coincidence of integer part. Indeed, the construction of $R^r(\mathcal{A})$ took ca. 74s., and the construction of $R(\mathcal{A} \times \mathcal{A})$ did not complete due to RAM shortage. Therefore we couldn't complete pre-processing for Boyer-Moore.

We note however that our naive algorithm worked fine. See Table 4 in Appendix G.

Case 5: An Automotive Example This final example (Fig. 11) is about anomaly detection of engines. The execution time is shown for 10 timed words w whose lengths range from 242,808 to 4,873,207. Each w is obtained as a discretized log of the simulation of the model `sldemo_enginewc.slx` in the Simulink Demo palette [18]: here the input of the model (desired rpm) is generated randomly according to the Gaussian distribution with $\mu = 2,000\text{rpm}$ and $\sigma^2 = 10^6\text{rpm}^2$; we discretized the output of the model (engine torque) into two statuses, *high* and *low*, with the threshold of $40\text{N} \cdot \text{m}$.

This test case is meant to be a practical example in automotive applications—our original motivation for the current work. The automaton \mathcal{A} expresses: the engine torque is *high* for more than 1s. (the kind of anomaly we are interested in) and the log is not too sparse (which means the log is a credible one).

Here the Boyer-Moore algorithm outperforms the naive one roughly by twice. The pre-processing phase took 9.94ms. on average.

Lacking in the current section are: detailed comparison with the existing implementations (e.g. in [22], modulo the word-signal difference in Rem. 2.2); and performance analysis when the specification \mathcal{A} , instead of the input timed word w , grows. We intend to address these issues in the coming extended version.

Acknowledgments Thanks are due to the anonymous referees for their careful reading and expert comments. The authors are supported by Grant-in-Aid No. 15KT0012, JSPS; T.A. is supported by Grant-in-Aid for JSPS Fellows.

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. E. Asarin, P. Caspi and O. Maler. A Kleene theorem for timed automata. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pp. 160–171. IEEE Computer Society, 1997.
3. E. Asarin, P. Caspi and O. Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.
4. E. Bartocci and R. Majumdar, editors. *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, vol. 9333 of *Lecture Notes in Computer Science*. Springer, 2015.
5. G. Behrmann, P. Bouyer, K.G. Larsen and R. Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006.
6. B. Bonakdarpour and S.A. Smolka, editors. *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, vol. 8734 of *Lecture Notes in Computer Science*. Springer, 2014.
7. R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.
8. C. Colombo and G.J. Pace. Fast-forward runtime monitoring - an industrial case study. In S. Qadeer and S. Tasiran, editors, *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, vol. 7687 of *Lecture Notes in Computer Science*, pp. 214–228. Springer, 2012.
9. J.V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal and S.A. Seshia. Robust online monitoring of signal temporal logic. In Bartocci and Majumdar [4], pp. 55–70.
10. A. Dokhanchi, B. Hoxha and G.E. Fainekos. On-line monitoring for temporal logic robustness. In Bonakdarpour and Smolka [6], pp. 231–246.
11. A. Donzé, T. Ferrère and O. Maler. Efficient robust monitoring for STL. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, vol. 8044 of *Lecture Notes in Computer Science*, pp. 264–279. Springer, 2013.
12. T. Ferrère, O. Maler, D. Nickovic and D. Ulus. Measuring with timed patterns. In D. Kroening and C.S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, vol. 9207 of *Lecture Notes in Computer Science*, pp. 322–337. Springer, 2015.
13. J. Geist, K.Y. Rozier and J. Schumann. Runtime observer pairs and bayesian network reasoners on-board fpgas: Flight-certifiable system health management for embedded systems. In Bonakdarpour and Smolka [6], pp. 215–230.
14. P. Herrmann. Renaming is necessary in timed regular expressions. In C.P. Rangan, V. Raman and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings*, vol. 1738 of *Lecture Notes in Computer Science*, pp. 47–59. Springer, 1999.
15. H. Ho, J. Ouaknine and J. Worrell. Online monitoring of metric temporal logic. In Bonakdarpour and Smolka [6], pp. 178–192.
16. A. Kane, O. Chowdhury, A. Datta and P. Koopman. A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In Bartocci and Majumdar [4], pp. 102–117.
17. D.E. Knuth, J.H.M. Jr. and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.
18. The MathWorks, Inc., Natick, MA, USA. *Simulink User’s Guide*, 2015.
19. Boyer-Moore Fast String Searching Example. <http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/fstrpos-example.html>.

20. J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.
21. P.K. Pandya and P.V. Suman. An introduction to timed automata. In *Modern Applications of Automata Theory*, pp. 111–148. World Scientific, 2012.
22. D. Ulus, T. Ferrère, E. Asarin and O. Maler. Timed pattern matching. In A. Legay and M. Bozga, editors, *Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings*, vol. 8711 of *Lecture Notes in Computer Science*, pp. 222–236. Springer, 2014.
23. D. Ulus, T. Ferrère, E. Asarin and O. Maler. Online timed pattern matching using derivatives. In M. Chechik and J. Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, vol. 9636 of *Lecture Notes in Computer Science*, pp. 736–751. Springer, 2016.
24. M. Waga, T. Akazaki and I. Hasuo. Code that Accompanies "A Boyer-Moore Type Algorithm for Timed Pattern Matching.". <https://github.com/MasWag/timed-pattern-matching>.
25. B.W. Watson and R.E. Watson. A boyer-moore-style algorithm for regular expression pattern matching. *Sci. Comput. Program.*, 48(2-3):99–117, 2003.

A Skip Value Functions in the (Original) Boyer-Moore Algorithm

Here $a \in \Sigma$ is a character, and $p \in \mathbb{Z}_{>0}$ is a positive integer.

$$\begin{aligned}\Delta_1(a, p) &= \min\{p - k \mid \text{pat}(k) = a \text{ or } k = 0\} \\ d_1(p) &= \min\{n \in \mathbb{Z}_{>0} \mid \text{pat}(p + 1 - n, |\text{pat}| - n) = \text{pat}(p + 1, |\text{pat}|) \text{ or } n \geq |\text{pat}|\} \\ d_2 &= \min\{n \in \mathbb{Z}_{>0} \mid \text{pat}(1, |\text{pat}| - n) = \text{pat}(n + 1, |\text{pat}|)\} \\ \Delta_2(p) &= \min\{d_1(p), d_2\}\end{aligned}$$

B Skip Value Function in the Boyer-Moore Type Algorithm for Pattern Matching

The definitions below follow those in [25].

Here s is a state of the automaton $\mathcal{A} = (\Sigma, S, S_0, E, F)$ that accepts the reversed language L^{Rev} of the given pattern L ; we let $\mathcal{A}_s = (\Sigma, S, S_0, E, \{s\})$ be the automaton where s is the only accepting state.

$$\begin{aligned}m_s &= \min\{|w| \mid w \in L(\mathcal{A}_s)\} \quad (\text{the length of a shortest word that leads to } s) \\ m &= \min_{s \in F} m_s \quad (\text{the length of a shortest accepted word}) \\ L' &= \{w(1, m)^{\text{Rev}} \mid w \in L(\mathcal{A})\} \quad (\text{for } w' \text{ to be accepted we need } w'(1, m) \in L') \\ L'_s &= \{w(1, \min\{m_s, m\})^{\text{Rev}} \mid w \in L(\mathcal{A}_s)\} \\ &\quad (\text{for } w' \text{ to lead to } s \text{ we need } w'(1, \min\{m_s, m\}) \in L'_s) \\ d_1(w) &= \min\{n \in \mathbb{Z}_{>0} \mid \Sigma^* w \Sigma^n \cap L' \neq \emptyset\} \\ d_2(w) &= \min\{n \in \mathbb{Z}_{>0} \mid w \Sigma^n \cap \Sigma^* L' \neq \emptyset\} \\ \Delta_2(s) &= \min_{w \in L'_s} \min\{d_1(w), d_2(w)\}\end{aligned}$$

For example Fig. 5 allows us to conclude that $d_2(\text{dc}) = 2$, and hence that $\Delta_2(s_3) = 2$.

Finally we define, for each $C \subseteq S$,

$$\Delta_2(C) = \max_{s \in C} \Delta_2(s) .$$

The other skip value function Δ_1 is defined as follows. For $a \in \Sigma$ and $p \in \mathbb{Z}_{>0}$,

$$\Delta_1(a, p) = \min\{p - k \mid k = 0 \text{ or } \exists w \in L'. w(k) = a\} .$$

C Further Details on Algorithm 1

Here are some further explanations on Algorithm 1.

- We separately compute: 1) those “immediately ending” matching intervals such that $w|_{(t, t')}$ is of length 1 (in which case only the terminal character $\$$ would occur); and 2) the others, i.e. such that there exists $i \in [1, |w|]$ with $\tau_i \in (t, t')$.

- Lines 2–5 compute a finite set $Immd$ of zones, such that $\bigcup Immd = \{(t, t') \mid \varepsilon|_{(t, t')} \in L(\mathcal{A})\}$. This is used as an overapproximation of the “immediately ending” matching intervals. Precisely: $\mathcal{M}(w, \mathcal{A}) \cap \{(t, t') \mid |w|_{(t, t')}| = 1\}$ coincides with $(\bigcup Immd) \cap \{(t, t') \mid \forall i \in [1, |w|], \tau_i \notin (t, t')\}$.
- For computing any other matching interval (t, t') —say $t \in [\tau_{i-1}, \tau_i]$ and $t' \in (\tau_j, \tau_{j+1}]$ —we use the following set.

$$Conf(i, j) = \{(s, \rho, T) \mid \forall t_0 \in T. \exists (\bar{s}, \bar{\nu}). (\bar{s}, \bar{\nu}) \text{ is a run over } w(i, j) - t_0, \\ s_{|\bar{s}|-1} = s, \text{ and } \nu_{|\bar{\nu}|-1} = \text{eval}(\rho, \tau_j, t_0)\}$$

Recall from (1) in Def. 2.4 that $s_{|\bar{s}|-1}$ is the last element of a sequence \bar{s} ; similarly for $\bar{\nu}$. The intuition of a “configuration” $(s, \rho, T) \in Conf(i, j)$ is: a nonempty interval T (that is a subset of $[\tau_{i-1}, \tau_i]$) is the set of epoch times t such that, after reading the timed word $w(i, j)$, we end up in a state $s \in S$ and have ρ as the record of clock reset. In the algorithm we use $CurrConf$ and $PrevConf$ for computing $Conf(i, j)$.

More specifically, we initialize T to be the whole $[\tau_{i-1}, \tau_i]$ (line 13). We gradually narrow it down, so that the relevant clock constraint is satisfied (line 18). Here we exploit our definition of clock constraint—we do not allow \neg or \vee —to ensure that the new set T' is indeed an interval.

On lines 25–31 we try to insert the terminal character $\$$ in $(\tau_j, \tau_{j+1}]$. It is also easy to see that $\text{solConstr}(T', T'', \rho', \delta')$ on line 31 is indeed a zone, due to the definition of a clock constraint δ' .

D An Online Variant of Our (Naive) Timed Pattern Matching Algorithm

An online variant of our naive algorithm (Alg. 1) is in Alg. 2. Here, unlike in Alg. 1, we do not divide the match set into “immediately ending” ones and others. The bottom line in this online algorithm is to compute $\mathcal{M}(w, \mathcal{A}) \cap \{(t, t') \mid t' \in (\tau_i, \tau_{i+1}]\}$ for each i . This algorithm is “online” because after reading a prefix $w(1, n)$ of w , we have the partial match set $\mathcal{M}(w, \mathcal{A}) \cap \{(t, t') \mid t' \in (\tau_0, \tau_n]\}$ at the point n . This algorithm consists of two main parts. In the former part—the main loop—we conduct the following procedure for each $i \in [1, |w|]$.

1. Lines 4–8 try to insert $\$$ in $(\tau_{i-1}, \tau_i]$.
2. Lines 10–18 read (a_i, τ_i) and build the set $\bigcup_{k \in [1, i]} Conf(k, i)$ of “configurations”.

In the latter part—post-processing—on lines 20–24, we try to insert $\$$ in $(\tau_{|w|}, \infty)$.

E Our Boyer-Moore Type Algorithm for Timed Pattern Matching

See Alg. 3.

Its main differences from the naive one (Alg. 1) are: 1) initially we start with $i = |w| - m + 1$ (line 1) instead of $i = |w|$ (line 1 of Alg. 1); and 2) we decrement i by the skip value computed by Δ_2 (line 33), instead of by 1 (line 33 of Alg. 1).

Algorithm 2 An online variant of our naive algorithm for timed pattern matching

Require: A timed word $w = (\bar{a}, \bar{\tau})$, and a timed automaton $\mathcal{A} = (\Sigma, S, S_0, C, E, F)$.

Ensure: $\bigcup Z$ is the match set $\mathcal{M}(w, \mathcal{A})$ in Def. 3.2.

```

1:  $CurrConf \leftarrow \emptyset$ ;  $Z \leftarrow \emptyset$ 
2: for  $i \leftarrow 1$  to  $|w|$  do ▷ Here  $CurrConf = \bigcup_{k \in [1, i-1]} Conf(k, i-1)$ .
3:    $CurrConf \leftarrow CurrConf \cup \{(s, \rho_\emptyset, [\tau_{i-1}, \tau_i]) \mid s \in S_0\}$ 
4:   for  $(s, \rho, T) \in CurrConf$  do
5:     for  $s_f \in F$  do
6:       for  $(s, s_f, \$, \lambda, \delta) \in E$  do
7:          $T' \leftarrow (\tau_{i-1}, \tau_i]$ 
8:          $Z \leftarrow Z \cup \text{solConstr}(T, T', \rho, \delta)$  ▷ Lines 4–8 try to insert  $\$$  in  $(\tau_{i-1}, \tau_i]$ .
9:    $(PrevConf, CurrConf) \leftarrow (CurrConf, \emptyset)$ 
10:  for  $(s, \rho, T) \in PrevConf$  do
11:    for  $(s, s', a_i, \lambda, \delta) \in E$  do ▷ Read  $(a_i, \tau_i)$ .
12:     $T' \leftarrow \{t_0 \in T \mid \text{eval}(\rho, \tau_i, t_0) \models \delta\}$ 
13:    ▷ Narrow the interval  $T$  to satisfy the clock constraint  $\delta$ .
14:    if  $T' \neq \emptyset$  then
15:       $\rho' \leftarrow \rho$ 
16:      for  $x \in \lambda$  do
17:         $\rho' \leftarrow \text{reset}(\rho', x, \tau_i)$  ▷ Reset the clock variables in  $\lambda$ .
18:       $CurrConf \leftarrow CurrConf \cup (s', \rho', T')$ 
19:   $CurrConf \leftarrow CurrConf \cup \{(s, \rho_\emptyset, [\tau_{|w|}, \infty)) \mid s \in S_0\}$ 
20:  for  $(s, \rho, T) \in CurrConf$  do
21:    for  $s_f \in F$  do
22:      for  $(s, s_f, \$, \lambda, \delta) \in E$  do
23:         $T' \leftarrow (\tau_{|w|}, \infty)$ 
24:         $Z \leftarrow Z \cup \text{solConstr}(T, T', \rho, \delta)$  ▷ Lines 20–24 try to insert  $\$$  in  $(\tau_{|w|}, \infty)$ .

```

Compared to Alg. 1 we have added lines 6–7, too. This is needed to take care of matching intervals in which no event occurs.

When one wishes to incorporate the other skip value function Δ_1 too, line 33 of Alg. 3 should be replaced by $i \leftarrow i - \max\{\Delta_1(a_j, j - i + 1), \Delta_2(CurrConf)\}$.

Algorithm 3 Our Boyer-Moore type algorithm for timed pattern matching

Require: A timed word $w = (\bar{a}, \bar{\tau})$, and a timed automaton $\mathcal{A} = (\Sigma, S, S_0, C, E, F)$.

Ensure: $\bigcup Z$ is the match set $\mathcal{M}(w, \mathcal{A})$ in Def. 3.2.

```

1:  $i \leftarrow |w| - m + 1$ ;  $CurrConf \leftarrow \emptyset$ ;  $Immd \leftarrow \emptyset$ ;  $Z \leftarrow \emptyset$ 
2: for  $s \in S_0$  do
3:   for  $s_f \in F$  do
4:     for  $(s, s_f, \$, \lambda, \delta) \in E$  do
5:        $Immd \leftarrow Immd \cup \text{solConstr}([0, \infty), (0, \infty), \rho_\emptyset, \delta)$ 
6: for  $k = 1$  to  $|w|$  do
7:    $Z \leftarrow Z \cup \{(T_0 \cap [\tau_{k-1}, \tau_k], T_f \cap (\tau_{k-1}, \tau_k], T_\Delta) \mid (T_0, T_f, T_\Delta) \in Immd\}$ 
8:    $Z \leftarrow Z \cup \{(T_0 \cap [\tau_{|w|}, \infty), T_f \cap (\tau_{|w|}, \infty), T_\Delta) \mid (T_0, T_f, T_\Delta) \in Immd\}$ 
9: while  $i > 0$  do
10:   $j \leftarrow i$ ;
11:   $CurrConf \leftarrow \{(s, \rho_\emptyset, [\tau_{i-1}, \tau_i]) \mid s \in S_0\}$ 
12:  while  $CurrConf \neq \emptyset \wedge j \leq |w|$  do
13:     $(PrevConf, CurrConf) \leftarrow (CurrConf, \emptyset)$ 
14:    for  $(s, \rho, T) \in PrevConf$  do
15:      for  $(s', s'_f, a_j, \lambda, \delta) \in E$  do
16:         $T' \leftarrow \{t_0 \in T \mid \delta(\text{eval}(\rho, \tau_j, t_0))\}$   $\triangleright$  Update available start interval.
17:        if  $T' \neq \emptyset$  then
18:           $\rho' \leftarrow \rho$ 
19:          for  $x \in \lambda$  do
20:             $\rho' \leftarrow \text{reset}(\rho', x, \tau_j)$   $\triangleright$  Reset clock variables in  $\lambda$ .
21:           $CurrConf \leftarrow CurrConf \cup (s', \rho', T')$ 
22:    for  $(s, \rho, T) \in CurrConf$  do
23:      for  $s_f \in F$  do
24:        for  $(s', s'_f, \$, \lambda', \delta') \in E$  do
25:          if  $j = |w|$  then
26:             $T'' \leftarrow (\tau_j, \infty)$ 
27:          else
28:             $T'' \leftarrow (\tau_j, \tau_{j+1}]$ 
29:           $Z \leftarrow Z \cup \text{solConstr}(T', T'', \rho', \delta')$   $\triangleright$  Solve the linear inequalities.
30:       $j \leftarrow j + 1$ 
31:    if  $j \leq |w|$  then
32:       $CurrConf \leftarrow PrevConf$ 
33:     $i \leftarrow i - \Delta_2(CurrConf)$ 

```

F Omitted Proofs

F.1 Proof of Thm. 4.3

Proof. For termination, it suffices to observe that line 31 of Alg. 1 is executed no more than $|w||E|^{|w|+1}$ times.

For correctness, assume first that there is a zone $(T_0, T_f, T_\Delta) \in Z$ such that $(t, t') \in (T_0, T_f, T_\Delta)$. For (T_0, T_f, T_Δ) , let \bar{e} be the sequence of transitions of \mathcal{A} visited by the

algorithm (line 17). This \bar{e} obviously yields an accepting run of \mathcal{A} over $w|_{(t,t')}$: line 18 ensures that all the clock constraints in \bar{e} are satisfied; and lines 25–31 ensure that the run is accepting.

Conversely, assume that there is an accepting run (\bar{s}, \bar{v}) of \mathcal{A} over $w|_{(t,t')}$. We extract a sequence \bar{e} of transitions from the run (\bar{s}, \bar{v}) ; and let (T_0, T_f, T_Δ) be the zone that is added to Z in the execution of Alg. 1 along the sequence \bar{e} (cf. line 17). Then it is straightforward to see that $(t, t') \in (T_0, T_f, T_\Delta)$. \square

F.2 Proof of Thm. 5.4

We consider a timed automaton $\mathcal{A} = (\Sigma \amalg \{\$, S, S_0, C, E, F)$ as the input of timed pattern matching.

Lemma F.1 *For any run $r = (\bar{s}, \bar{\alpha})$ of the region automata $R(\mathcal{A})$, the following equation holds.*

$$\text{pref}(\mathcal{W}(r)) = \mathcal{W}(\text{pref}(r))$$

Proof. Let $w \in \text{pref}(\mathcal{W}(r))$ and w' be a timed word such that $w \cdot w' \in \mathcal{W}(r)$. Let $(\bar{s}, \bar{v}) \in (\bar{s}, \bar{\alpha})$ be a run over $w \cdot w'$. Since $(\bar{s}(0, |w|), \bar{v}(0, |w|)) \in (\bar{s}(0, |w|), \bar{\alpha}(0, |w|))$ is a run over w over \mathcal{A} , w is a member of $\mathcal{W}(\text{pref}(r))$.

Let $w \in \mathcal{W}(\text{pref}(r))$ and n be an index of r such that $w \in \mathcal{W}(\bar{s}(0, n), \bar{\alpha}(0, n))$. Let $(\bar{s}(0, n), \bar{v})$ be a run of \mathcal{A} over w such that for any $0 \leq i \leq n$, $\bar{v}_i \in \bar{\alpha}_i$ holds. Since $(\bar{s}, \bar{\alpha})$ is a run of $R(\mathcal{A})$, there is a sequence of interpretations $\bar{\nu}'$ such that $(\bar{s}, \bar{v} \cdot \bar{\nu}') \in (\bar{s}, \bar{\alpha})$. Since $(\bar{s}, \bar{v} \cdot \bar{\nu}')$ is a run of \mathcal{A} , there is a timed word w' such that $(\bar{s}, \bar{v} \cdot \bar{\nu}')$ is a run over $w \cdot w'$. Thus, w is a member of $\text{pref}(\mathcal{W}(r))$.

Lemma F.2 *Let r and r' be runs of $R(\mathcal{A})$. For any integer $1 \leq n < |r'|$, the following property holds.*

$$(\Sigma^n \times (\mathbb{R}_{>0})^n) \cdot \mathcal{W}(r) \cap \mathcal{W}(r') \neq \emptyset \Rightarrow \mathcal{W}(r) \cap \mathcal{W}(r'(n, |r'|)) \neq \emptyset$$

Proof. Assume there is a timed word $(\bar{a}, \bar{\tau})$ over $(\Sigma \amalg \{\$, \mathbb{R}_{>0})^n$ such that $(\bar{a}, \bar{\tau}) \in (\Sigma \times (\mathbb{R}_{>0})^n) \cdot \mathcal{W}(r) \cap \mathcal{W}(r')$. By definition of non-absorbing concatenations, $(\bar{a}(n+1, |\bar{a}|), \bar{\tau}(n+1, |\bar{\tau}|)) - \tau_n \in \mathcal{W}(r)$ holds. Since $(\bar{a}, \bar{\tau}) \in \mathcal{W}(r')$, $(\bar{a}(n+1, |\bar{a}|), \bar{\tau}(n+1, |\bar{\tau}|)) - \tau_n \in \mathcal{W}(r'(n, |r'|))$ holds. Thus $(\bar{a}(n+1, |\bar{a}|), \bar{\tau}(n+1, |\bar{\tau}|)) - \tau_n$ is an element of $\mathcal{W}(r) \cap \mathcal{W}(r'(n, |r'|))$.

We define the *pre-accepted language* $L_{-\$}(\mathcal{A}) = \{w(1, |w| - 1) \mid w \in L(\mathcal{A})\}$. Since we consider timed pattern matching, the removed event of $L_{-\$}$ is $\$$. The language $L_{-\$}$ represent the pattern without the terminate character. By definition of L' , $L_{-\$}(\mathcal{A}) \subseteq \bigcup_{r \in L'} \mathcal{W}(r) \cdot (\Sigma \times \mathbb{R}_{>0})^*$ holds. For each $s \in S$, we define the language of words leading to s as $L_s = L((\Sigma \amalg \{\$, S, S_0, C, E, \{s\}))$.

Lemma F.3 *For any $s \in S \setminus F$, we have the following property.*

$$L_s \subseteq \bigcup_{r \in L'_s} \mathcal{W}(r) \cdot (\Sigma \times \mathbb{R}_{>0})^*$$

Proof. Let $w \in L_s$, and $(\bar{s}, \bar{\nu})$ be a corresponding run of w satisfying $s_{|\bar{s}|-1} = s$. We have

$$\begin{aligned} (\bar{s}, [\bar{\nu}]) &\in \{r \mid \beta_0 \in S_0^r, \beta \in R^r(s), r \in \text{Run}_{R^r(\mathcal{A})}(\beta_0, \beta)\} \\ &\subseteq \{r(0, \min\{m, m'_s\} - 1) \mid \beta_0 \in S_0^r, \beta \in R^r(s), r \in \text{Run}_{R^r(\mathcal{A})}(\beta_0, \beta)\} \cdot (S^r)^* \\ &= L'_s \cdot (S^r)^* \end{aligned}$$

where $[\bar{\nu}] = [\nu_0], [\nu_1], \dots, [\nu_{|\bar{\nu}|-1}]$. Thus, $w \in \mathcal{W}(\bar{s}, [\bar{\nu}]) \subseteq \bigcup_{r \in L'_s} \mathcal{W}(r) \cdot (\Sigma \times \mathbb{R}_{>0})^*$ holds.

For any i, j such that $1 \leq i < j \leq |w|$, for any $(s, \rho, T) \in \text{Conf}(i, j)$, we have $s \notin F$. The proof of the theorem. 5.4 is as follows.

Proof.

$$\begin{aligned} &\exists t \in [\tau_{i-n-1}, \tau_{i-n}). \exists t' \in (t, \infty). (t, t') \in \mathcal{M}(w, \mathcal{A}) \\ &\Leftrightarrow \exists t \in [\tau_{i-n-1}, \tau_{i-n}). \exists t' \in (t, \infty). w|_{(t, t')} \in L(\mathcal{A}) \\ &\Rightarrow \exists t \in [\tau_{i-n-1}, \tau_{i-n}). \exists t' \in (t, \infty). \exists k \in [i-n, |w|]. (w(i-n, k) - t) \circ (\$, t') \in L(\mathcal{A}) \\ &\Rightarrow \exists t \in [\tau_{i-n-1}, \tau_{i-n}). \exists k \in [i-n, |w|]. (w(i-n, k) - t) \in L_{-\$}(\mathcal{A}) \\ &\Rightarrow \exists t \in [\tau_{i-n-1}, \tau_{i-n}). (w(i-n, |w|) - t) \in L_{-\$}(\mathcal{A}) \cdot (\Sigma \times \mathbb{R}_{>0})^* \\ &\Rightarrow (\Sigma \times \mathbb{R}_{>0})^n \cdot (w(i, |w|) - \tau_{i-1}) \cap L_{-\$}(\mathcal{A}) \cdot (\Sigma \times \mathbb{R}_{>0})^* \neq \emptyset \\ &\Rightarrow (\Sigma \times \mathbb{R}_{>0})^n \cdot (w(i, j) - \tau_{i-1}) \cdot (\Sigma \times \mathbb{R}_{>0})^* \cap L_{-\$}(\mathcal{A}) \cdot (\Sigma \times \mathbb{R}_{>0})^* \neq \emptyset \\ &\Rightarrow \forall (s, \rho, T) \in \text{Conf}(i, j). (\Sigma \times \mathbb{R}_{>0})^n \cdot L_s \cdot (\Sigma \times \mathbb{R}_{>0})^* \cap L_{-\$}(\mathcal{A}) \cdot (\Sigma \times \mathbb{R}_{>0})^* \neq \emptyset \\ &\Rightarrow \forall (s, \rho, T) \in \text{Conf}(i, j). \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \cdot (\Sigma \times \mathbb{R}_{>0})^* \right) \cap \left(\bigcup_{r' \in L'} \mathcal{W}(r') \cdot (\Sigma \times \mathbb{R}_{>0})^* \right) \neq \emptyset \\ &\Leftrightarrow \forall (s, \rho, T) \in \text{Conf}(i, j). \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \cdot (\Sigma \times \mathbb{R}_{>0})^* \right) \cap \bigcup_{r' \in L'} \mathcal{W}(r') \neq \emptyset \\ &\vee \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \mathcal{W}(r') \cdot (\Sigma \times \mathbb{R}_{>0})^* \right) \neq \emptyset \\ &\Leftrightarrow \forall (s, \rho, T) \in \text{Conf}(i, j). \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \bigcup_{r'' \in \text{pref}(r')} \mathcal{W}(r'') \right) \neq \emptyset \\ &\vee \left(\bigcup_{r \in L'_s} \text{pref}((\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r)) \right) \cap \bigcup_{r' \in L'} \mathcal{W}(r') \neq \emptyset \end{aligned}$$

Thus,

$$\begin{aligned} &\text{Opt}(i) = \min\{n \in \mathbb{Z}_{>0} \mid \exists t \in [\tau_{i-n-1}, \tau_{i-n}), t' \in (t, \infty). (t, t') \in \mathcal{M}(w, \mathcal{A})\} \\ &\geq \min\{n \in \mathbb{Z}_{>0} \mid \forall (s, \rho, T) \in \text{Conf}(i, j). \\ &\left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \bigcup_{r'' \in \text{pref}(r')} \mathcal{W}(r'') \right) \neq \emptyset \\ &\vee \left(\bigcup_{r \in L'_s} \text{pref}((\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r)) \right) \cap \bigcup_{r' \in L'} \mathcal{W}(r') \neq \emptyset\} \end{aligned}$$

$$\begin{aligned}
& \{\text{Such } n \text{ is not more than } m\} \\
&= \min\{n \in \mathbb{Z}_{>0} \mid \forall (s, \rho, T) \in \text{Conf}(i, j). \\
& \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \bigcup_{r'' \in \text{pref}(r')} \mathcal{W}(r'') \right) \neq \emptyset \\
& \vee \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \text{pref}(\mathcal{W}(r)) \right) \cap \bigcup_{r' \in L'} \mathcal{W}(r') \neq \emptyset \} \\
&= \min\{n \in \mathbb{Z}_{>0} \mid \forall (s, \rho, T) \in \text{Conf}(i, j). \\
& \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \bigcup_{r'' \in \text{pref}(r')} \mathcal{W}(r'') \right) \neq \emptyset \\
& \vee \left(\bigcup_{r \in L'_s} \bigcup_{r'' \in \text{pref}(r)} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r'') \right) \cap \bigcup_{r' \in L'} \mathcal{W}(r') \neq \emptyset \} \\
& \{\text{Conf}(i, j) \text{ is finite.}\} \\
&= \max_{(s, \rho, T) \in \text{Conf}(i, j)} \min\{n \in \mathbb{Z}_{>0} \mid \\
& \left(\bigcup_{r \in L'_s} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \bigcup_{r'' \in \text{pref}(r')} \mathcal{W}(r'') \right) \neq \emptyset \\
& \vee \left(\bigcup_{r \in L'_s} \bigcup_{r'' \in \text{pref}(r)} (\Sigma \times \mathbb{R}_{>0})^n \cdot \mathcal{W}(r'') \right) \cap \bigcup_{r' \in L'} \mathcal{W}(r') \neq \emptyset \} \\
& \geq \max_{(s, \rho, T) \in \text{Conf}(i, j)} \min\{n \in \mathbb{Z}_{>0} \mid \\
& \left(\bigcup_{r \in L'_s} \mathcal{W}(r) \right) \cap \left(\bigcup_{r' \in L'} \bigcup_{r'' \in \text{pref}(r'(n, |r'|))} \mathcal{W}(r'') \right) \neq \emptyset \\
& \vee \left(\bigcup_{r \in L'_s} \bigcup_{r'' \in \text{pref}(r)} \mathcal{W}(r'') \right) \cap \left(\bigcup_{r' \in L'} \mathcal{W}(r'(n, |r'|)) \right) \neq \emptyset \} \\
&= \max_{(s, \rho, T) \in \text{Conf}(i, j)} \min_{r \in L'_s} \min\{ \min_{r' \in L'} \min\{n \in \mathbb{Z}_{>0} \mid \mathcal{W}(r) \cap \left(\bigcup_{r'' \in \text{pref}(r'(n, |r'|))} \mathcal{W}(r'') \right) \neq \emptyset \}, \\
& \min_{r' \in L'} \min\{n \in \mathbb{Z}_{>0} \mid \left(\bigcup_{r'' \in \text{pref}(r)} \mathcal{W}(r'') \right) \cap \mathcal{W}(r'(n, |r'|)) \neq \emptyset \} \} \\
&= \max_{(s, \rho, T) \in \text{Conf}(i, j)} \min_{r \in L'_s} \min\{d_1(r), d_2(r)\} \\
&= \Delta_2(\text{Conf}(i, j))
\end{aligned}$$

F.3 Proof of Prop. 5.5

Proof. Assume there is a timed word $(\bar{a}, \bar{\tau})$ such that $(\bar{a}, \bar{\tau}) \in \mathcal{W}(r) \cap \mathcal{W}(r')$. Let $(\bar{s}, \bar{\nu}) \in r$ be a partial run of \mathcal{A} over $(\bar{a}, \bar{\tau})$ and $(\bar{s}', \bar{\nu}') \in r'$ be a partial run of \mathcal{A}' over $(\bar{a}, \bar{\tau})$. Since $((\bar{s}, \bar{s}'), (\bar{\nu}, \bar{\nu}')) \in (r, r')$ is a partial run over $(\bar{a}, \bar{\tau})$ of $\mathcal{A} \times \mathcal{A}'$, $(\bar{a}, \bar{\tau})$ is a member of $\mathcal{W}((r, r'))$.

Assume there is a timed word $(\bar{a}, \bar{\tau})$ such that $(\bar{a}, \bar{\tau}) \in \mathcal{W}((r, r'))$. Let $((\bar{s}, \bar{s}'), (\bar{v}, \bar{v}')) \in (r, r')$ be a partial run over $(\bar{a}, \bar{\tau})$ of $\mathcal{A} \times \mathcal{A}'$ where \bar{v} is a clock interpretation over C and \bar{v}' is a clock interpretation over C' . Since $(\bar{s}, \bar{v}) \in r$ is a partial run over $(\bar{a}, \bar{\tau})$ of \mathcal{A} and $(\bar{s}', \bar{v}') \in r'$ is a partial run over $(\bar{a}, \bar{\tau})$ over \mathcal{A}' , $(\bar{a}, \bar{\tau}) \in \mathcal{W}(r) \cap \mathcal{W}(r')$.

G Detailed Results of The Experiments

The detailed results of our experiments are in the following tables.

Table 1. Execution time in Case 1.

$ w $	Naive Time (ms.)	BM Time (ms.)	BM + Preproc. Time (ms.)	$ Z $
20	$4.65 \cdot 10^{-3}$	$7.62 \cdot 10^{-3}$	$8.22 \cdot 10^{-2}$	10
40	$7.77 \cdot 10^{-3}$	$6.56 \cdot 10^{-3}$	$5.38 \cdot 10^{-2}$	20
80	$1.81 \cdot 10^{-2}$	$1.34 \cdot 10^{-2}$	$6.11 \cdot 10^{-2}$	40
160	$3.53 \cdot 10^{-2}$	$4.43 \cdot 10^{-2}$	$1.26 \cdot 10^{-1}$	80
320	$4.52 \cdot 10^{-2}$	$4.29 \cdot 10^{-2}$	$9.24 \cdot 10^{-2}$	160
640	$1.20 \cdot 10^{-1}$	$8.18 \cdot 10^{-2}$	$1.39 \cdot 10^{-1}$	320
1,000	$1.75 \cdot 10^{-1}$	$1.47 \cdot 10^{-1}$	$2.19 \cdot 10^{-1}$	500
1,280	$2.36 \cdot 10^{-1}$	$1.77 \cdot 10^{-1}$	$2.36 \cdot 10^{-1}$	640
2,000	$2.44 \cdot 10^{-1}$	$3.00 \cdot 10^{-1}$	$3.78 \cdot 10^{-1}$	1,000
2,560	$3.67 \cdot 10^{-1}$	$2.88 \cdot 10^{-1}$	$3.42 \cdot 10^{-1}$	1,280
3,000	$3.48 \cdot 10^{-1}$	$3.22 \cdot 10^{-1}$	$3.76 \cdot 10^{-1}$	1,500
4,000	$3.90 \cdot 10^{-1}$	$4.22 \cdot 10^{-1}$	$4.80 \cdot 10^{-1}$	2,000
5,000	$5.54 \cdot 10^{-1}$	$6.89 \cdot 10^{-1}$	$7.58 \cdot 10^{-1}$	2,500
5,120	$7.38 \cdot 10^{-1}$	$5.78 \cdot 10^{-1}$	$6.35 \cdot 10^{-1}$	2,560
6,000	$1.09 \cdot 10^0$	$7.87 \cdot 10^{-1}$	$8.59 \cdot 10^{-1}$	3,000
7,000	$1.21 \cdot 10^0$	$7.97 \cdot 10^{-1}$	$8.60 \cdot 10^{-1}$	3,500
8,000	$1.05 \cdot 10^0$	$7.06 \cdot 10^{-1}$	$7.57 \cdot 10^{-1}$	4,000
9,000	$1.40 \cdot 10^0$	$1.19 \cdot 10^0$	$1.26 \cdot 10^0$	4,500
10,000	$1.86 \cdot 10^0$	$1.02 \cdot 10^0$	$1.07 \cdot 10^0$	5,000
10,240	$1.43 \cdot 10^0$	$1.08 \cdot 10^0$	$1.14 \cdot 10^0$	5,120
16,000	$2.57 \cdot 10^0$	$2.24 \cdot 10^0$	$2.32 \cdot 10^0$	8,000
32,000	$4.60 \cdot 10^0$	$3.64 \cdot 10^0$	$3.71 \cdot 10^0$	16,000
64,000	$6.85 \cdot 10^0$	$6.82 \cdot 10^0$	$6.89 \cdot 10^0$	32,000
100,000	$1.00 \cdot 10^1$	$8.88 \cdot 10^0$	$8.95 \cdot 10^0$	50,000
128,000	$1.16 \cdot 10^1$	$1.03 \cdot 10^1$	$1.04 \cdot 10^1$	64,000
200,000	$1.80 \cdot 10^1$	$1.61 \cdot 10^1$	$1.61 \cdot 10^1$	100,000
256,000	$2.19 \cdot 10^1$	$1.93 \cdot 10^1$	$1.94 \cdot 10^1$	128,000
300,000	$2.80 \cdot 10^1$	$2.51 \cdot 10^1$	$2.52 \cdot 10^1$	150,000
400,000	$3.49 \cdot 10^1$	$3.11 \cdot 10^1$	$3.11 \cdot 10^1$	200,000
500,000	$4.20 \cdot 10^1$	$3.74 \cdot 10^1$	$3.75 \cdot 10^1$	250,000
512,000	$4.35 \cdot 10^1$	$3.76 \cdot 10^1$	$3.77 \cdot 10^1$	256,000
600,000	$5.60 \cdot 10^1$	$4.85 \cdot 10^1$	$4.85 \cdot 10^1$	300,000
700,000	$6.14 \cdot 10^1$	$5.46 \cdot 10^1$	$5.47 \cdot 10^1$	350,000
800,000	$6.81 \cdot 10^1$	$6.02 \cdot 10^1$	$6.03 \cdot 10^1$	400,000
900,000	$7.51 \cdot 10^1$	$6.63 \cdot 10^1$	$6.64 \cdot 10^1$	450,000
1,000,000	$8.29 \cdot 10^1$	$7.19 \cdot 10^1$	$7.20 \cdot 10^1$	500,000
1,024,000	$8.41 \cdot 10^1$	$7.39 \cdot 10^1$	$7.40 \cdot 10^1$	512,000

Table 2. Execution time in Case 2.

$ w $	Naive Time (ms.)	BM Time (ms.)	BM + Preproc. Time (ms.)	$ Z $
20	$1.80 \cdot 10^{-2}$	$4.80 \cdot 10^{-2}$	$1.59 \cdot 10^2$	0
40	$6.03 \cdot 10^{-2}$	$6.27 \cdot 10^{-1}$	$1.37 \cdot 10^2$	0
80	$3.06 \cdot 10^{-1}$	$7.80 \cdot 10^{-1}$	$1.35 \cdot 10^2$	0
160	$1.11 \cdot 10^0$	$1.45 \cdot 10^0$	$1.31 \cdot 10^2$	0
320	$4.20 \cdot 10^0$	$4.92 \cdot 10^0$	$1.37 \cdot 10^2$	0
640	$1.66 \cdot 10^1$	$1.64 \cdot 10^1$	$1.38 \cdot 10^2$	0
1,000	$6.74 \cdot 10^1$	$3.83 \cdot 10^1$	$1.88 \cdot 10^2$	0
1,280	$8.82 \cdot 10^1$	$6.18 \cdot 10^1$	$2.04 \cdot 10^2$	0
2,000	$1.71 \cdot 10^2$	$1.49 \cdot 10^2$	$2.98 \cdot 10^2$	0
2,560	$2.49 \cdot 10^2$	$2.49 \cdot 10^2$	$3.87 \cdot 10^2$	0
3,000	$3.46 \cdot 10^2$	$3.36 \cdot 10^2$	$4.63 \cdot 10^2$	0
4,000	$5.73 \cdot 10^2$	$5.92 \cdot 10^2$	$7.46 \cdot 10^2$	1
5,000	$8.48 \cdot 10^2$	$9.26 \cdot 10^2$	$1.07 \cdot 10^3$	0
5,120	$9.10 \cdot 10^2$	$1.01 \cdot 10^3$	$1.13 \cdot 10^3$	0
6,000	$1.25 \cdot 10^3$	$1.36 \cdot 10^3$	$1.52 \cdot 10^3$	0
7,000	$1.66 \cdot 10^3$	$1.84 \cdot 10^3$	$1.97 \cdot 10^3$	0
8,000	$2.16 \cdot 10^3$	$2.41 \cdot 10^3$	$2.55 \cdot 10^3$	0
9,000	$2.72 \cdot 10^3$	$3.06 \cdot 10^3$	$3.19 \cdot 10^3$	0
10,000	$3.35 \cdot 10^3$	$3.73 \cdot 10^3$	$3.88 \cdot 10^3$	1
10,240	$3.54 \cdot 10^3$	$3.95 \cdot 10^3$	$4.08 \cdot 10^3$	0

Table 3. Execution time in Case 3.

λ	$\tau_{ \overline{\tau} }$	$ w $	Naive Time (ms.)	BM Time (ms.)	BM + Preproc. Time (ms.)	$ Z $
0.20	$4.00 \cdot 10^4$	8,028	0.83	0.54	$1.02 \cdot 10^1$	54
0.20	$8.00 \cdot 10^4$	15,958	1.68	0.84	$1.10 \cdot 10^1$	58
0.20	$1.60 \cdot 10^5$	31,826	2.68	1.35	$1.09 \cdot 10^1$	105
0.20	$3.20 \cdot 10^5$	64,026	5.84	1.91	$9.06 \cdot 10^0$	331
0.20	$3.20 \cdot 10^6$	639,762	49.26	14.59	$2.04 \cdot 10^1$	2,410
0.20	$6.40 \cdot 10^6$	1,279,143	98.51	28.75	$3.45 \cdot 10^1$	5,012
0.20	$1.28 \cdot 10^7$	2,559,898	198.45	58.10	$6.41 \cdot 10^1$	10,602
0.30	$4.00 \cdot 10^4$	11,898	1.53	0.81	$1.18 \cdot 10^1$	124
0.30	$8.00 \cdot 10^4$	23,739	3.32	1.34	$1.22 \cdot 10^1$	176
0.30	$1.60 \cdot 10^5$	48,202	4.74	1.71	$9.07 \cdot 10^0$	370
0.30	$3.20 \cdot 10^5$	96,195	7.74	2.72	$8.73 \cdot 10^0$	948
0.30	$3.20 \cdot 10^6$	960,292	75.42	25.97	$3.19 \cdot 10^1$	9,267
0.30	$6.40 \cdot 10^6$	1,921,414	151.25	50.61	$5.64 \cdot 10^1$	18,385
0.30	$1.28 \cdot 10^7$	3,836,454	303.69	101.04	$1.07 \cdot 10^2$	36,788
0.50	$4.00 \cdot 10^4$	19,991	3.13	1.38	$1.09 \cdot 10^1$	532
0.50	$8.00 \cdot 10^4$	40,107	4.89	2.18	$1.07 \cdot 10^1$	866
0.50	$1.60 \cdot 10^5$	79,965	7.71	3.47	$1.05 \cdot 10^1$	1,678
0.50	$3.20 \cdot 10^5$	159,771	13.19	5.81	$1.17 \cdot 10^1$	3,391
0.50	$3.20 \cdot 10^6$	1,599,831	136.57	56.25	$6.20 \cdot 10^1$	35,275
0.50	$6.40 \cdot 10^6$	3,200,371	268.47	112.53	$1.18 \cdot 10^2$	72,153
0.50	$1.28 \cdot 10^7$	6,400,401	534.35	223.59	$2.29 \cdot 10^2$	$1.43 \cdot 10^5$
0.60	$4.00 \cdot 10^4$	23,731	2.83	1.78	$1.17 \cdot 10^1$	663
0.60	$8.00 \cdot 10^4$	47,953	5.42	2.84	$1.11 \cdot 10^1$	1,280
0.60	$1.60 \cdot 10^5$	96,414	9.13	4.32	$1.09 \cdot 10^1$	2,686
0.60	$3.20 \cdot 10^5$	192,552	16.50	7.81	$1.36 \cdot 10^1$	5,310
0.60	$3.20 \cdot 10^6$	1,920,929	163.81	75.15	$8.10 \cdot 10^1$	53,044
0.60	$6.40 \cdot 10^6$	3,838,606	325.40	149.19	$1.55 \cdot 10^2$	$1.06 \cdot 10^5$
0.60	$1.28 \cdot 10^7$	7,684,548	653.87	298.21	$3.04 \cdot 10^2$	$2.14 \cdot 10^5$
0.80	$4.00 \cdot 10^4$	32,310	3.58	2.80	$1.28 \cdot 10^1$	1,099
0.80	$8.00 \cdot 10^4$	64,368	7.34	3.86	$1.09 \cdot 10^1$	2,207
0.80	$1.60 \cdot 10^5$	127,930	11.70	6.28	$1.22 \cdot 10^1$	4,488
0.80	$3.20 \cdot 10^5$	255,611	22.77	12.22	$1.80 \cdot 10^1$	8,331
0.80	$3.20 \cdot 10^6$	2,560,634	231.75	120.68	$1.27 \cdot 10^2$	86,511
0.80	$6.40 \cdot 10^6$	5,121,063	448.66	237.66	$2.44 \cdot 10^2$	$1.73 \cdot 10^5$
0.80	$1.28 \cdot 10^7$	10,243,600	892.60	478.18	$4.84 \cdot 10^2$	$3.45 \cdot 10^5$
1.00	$4.00 \cdot 10^4$	40,410	4.78	3.30	$1.22 \cdot 10^1$	1,112
1.00	$8.00 \cdot 10^4$	79,878	7.58	5.58	$1.36 \cdot 10^1$	2,137
1.00	$1.60 \cdot 10^5$	160,213	13.28	7.77	$1.39 \cdot 10^1$	2,683
1.00	$3.20 \cdot 10^5$	319,948	22.83	13.56	$1.93 \cdot 10^1$	2,793
1.20	$4.00 \cdot 10^4$	48,271	5.34	3.23	$9.93 \cdot 10^0$	1,224
1.20	$8.00 \cdot 10^4$	95,962	9.84	5.85	$1.24 \cdot 10^1$	2,313
1.20	$1.60 \cdot 10^5$	191,733	14.59	9.77	$1.57 \cdot 10^1$	2,947
1.20	$3.20 \cdot 10^5$	384,298	26.80	18.11	$2.39 \cdot 10^1$	3,070
1.50	$4.00 \cdot 10^4$	59,766	6.44	5.17	$1.34 \cdot 10^1$	1,157
1.50	$8.00 \cdot 10^4$	119,910	11.10	8.36	$1.51 \cdot 10^1$	2,241
1.50	$1.60 \cdot 10^5$	239,409	18.07	13.80	$1.97 \cdot 10^1$	2,804
2.00	$4.00 \cdot 10^4$	79,898	9.11	7.26	$1.52 \cdot 10^1$	846
2.00	$8.00 \cdot 10^4$	159,319	13.40	10.88	$1.69 \cdot 10^1$	1,610
2.00	$1.60 \cdot 10^5$	321,191	23.38	19.04	$2.49 \cdot 10^1$	1,964

Table 4. Execution time in Case 4.

λ	$\tau_{ \overline{\tau} }$	$ w $	Naive Time (ms.)	$ Z $
0.03	40,000	1,934	1.90	2,137
0.03	80,000	3,987	4.53	4,417
0.03	160,000	7,978	6.39	8,848
0.05	40,000	4,007	2.61	4,754
0.05	80,000	8,139	2.65	9,583
0.05	160,000	15,934	8.00	18,875
0.08	40,000	6,110	2.54	7,699
0.08	80,000	12,158	5.20	15,195
0.08	160,000	23,986	8.83	29,744
0.10	40,000	7,947	5.97	10,410
0.10	80,000	15,867	6.54	20,822
0.10	160,000	31,935	16.73	41,414

Table 5. Execution time in Case 5.

$ w $	Naive Time (ms.)	BM Time (ms.)	BM + Preproc. Time (ms.)	$ Z $
242,808	$1.48 \cdot 10^1$	$7.99 \cdot 10^0$	$1.81 \cdot 10^1$	97
487,021	$2.98 \cdot 10^1$	$1.58 \cdot 10^1$	$2.55 \cdot 10^1$	112
732,281	$4.46 \cdot 10^1$	$2.40 \cdot 10^1$	$3.37 \cdot 10^1$	112
1,221,149	$7.47 \cdot 10^1$	$3.91 \cdot 10^1$	$4.88 \cdot 10^1$	166
1,830,434	$1.15 \cdot 10^2$	$6.06 \cdot 10^1$	$7.08 \cdot 10^1$	216
2,436,963	$1.52 \cdot 10^2$	$7.79 \cdot 10^1$	$8.77 \cdot 10^1$	230
3,045,927	$1.88 \cdot 10^2$	$9.76 \cdot 10^1$	$1.08 \cdot 10^2$	230
3,654,904	$2.27 \cdot 10^2$	$1.17 \cdot 10^2$	$1.27 \cdot 10^2$	274
4,265,044	$2.60 \cdot 10^2$	$1.38 \cdot 10^2$	$1.48 \cdot 10^2$	336
4,873,207	$2.99 \cdot 10^2$	$1.58 \cdot 10^2$	$1.69 \cdot 10^2$	372